RESEARCH PAPER | Engineering | Volume : 5 | Issue : 2 | Feb 2015 | ISSN - 2249-555X

A Proposed Relan Algorithm for Booting Process in Regional Language for Linux Based Mobile Operating System

KEYWORDS: Linux, booting process, regional Language, Mobile Technology, Operating System, RELAN.

Mr. Milan S. Bhatt
Programmer, M. D. Gramseva Mahavidyalay, Gujarat Vidyapith.

Dr. Prashant M. Dolia
Associate Professor, Maharaja Krishnakumar Sinhji Bhavnagar Univeristy, Bhavnagar

ABSTRACT *The Information Technology, no any result for booting process in mobile technology with the use of regional language. Information technology without doubt, has enormous power to improve how people live and work. Thousands of tools – hardware, software, and embedded – are developed to make life of mankind an efficient and convenient for mobile technology. A revolution is taking place today in the way of people, how to access, learn, and interact with information looking booting process in regional language for the Linux based mobile Operating System. Research work is required to develop more assistive system in areas like proposed regional language algorithm for booting process in Linux based Operating System.*

1 Introduction

Most all the people are working with the mobile technology in different way. Specific people are uses the different mobile with specific operating system like Android, Windows, Asha, and Blackberry etc. But now a days almost booting process of operating systems in English Language.

So, ruler area people can't understand English language properly at starting of mobile at first time, So new proposed booting process and its component and its process (different messages) in regional language.

At the starting of the configuration of a new Mobile devices, mobile operating system's gives the message to the user in English language. So, specific domain regional language people may not be understand the different messages which is given by the mobile operating system. So, regional language people are easily understand the all the messages which is given by the mobile, transferring in regional language.

2 Language Processor for Booting Processing

Transferring the booting process of mobile devices English language to regional language booting process need some language processor for representation of an algorithm in a source language to and produces as output of target regional language.

2.1 Assemblers: It is used to translate the program written in Assembly language into machine code. An assembler performs the translation process in similar way as compiler. But assembler is the translator program for low-level programming language, while a compiler is the translator program for high-level programming languages.

2.2 Compilers: Language processors that map high-level language instructions into machine code, e.g. Delphi, GCC, Visual C++ etc. Implementation language and the source text to be compiled is actually a new version of the language processor itself, the process is called bootstrapping. The compilation of the compiler itself does not need to be done on the target machine, but instead it can take place on another machine; this is called cross-compilation.

2.3 Pre-Processors: Language processors that map a superset of high-level language into the original high-level language, or perform simple text substitutions before translation takes place.

2.4 Interpreters: Language processors that include an execution component, i.e. they perform the operations specified in the source text, rather than re- expressing them in another language; e.g. Matlab

2.5 Disassemblers: Language processors that attempt to take object code at a low level and regenerate source code at a higher level.

3 Language Translator

A language translator is a computer program that converts a program written in a procedural language into machine language that can be directly executed by the computer. Computers can execute only machine language programs. Programs written in any other language must be translated into a machine language load module, which is suitable for loading directly into primary storage.

The program translation model bridges the execution gap by translating a program written in a PL, called the source program (SP), into an equivalent program in the machine or assembly language of the computer system, called the target program (TP).

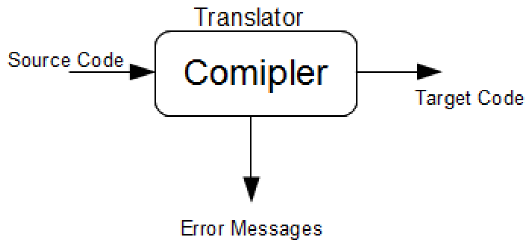
A program must be translated before it can be executed.

- The translated program may be saved in a file. The saved program may be executed repeatedly.

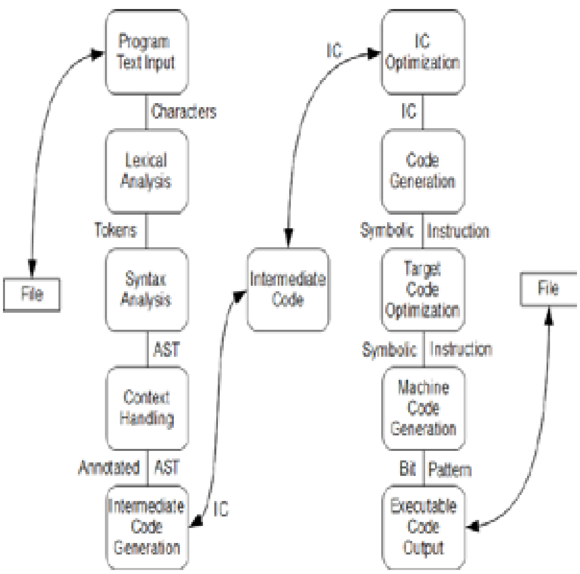
- A program must be re translated following modifications.

Language Processing = Analysis of Source Programme + Synthesis of Target Programme

Volume : 5 | Issue : 2 | Feb 2015 | ISSN - 2249-555X



[Fig. - 1 Language Processing]



[Fig. - 2 Conceptual Structure of Language Processing]

4 Gujarati Morphology

Identification, analysis and description of the structure of words. Study of structural variations of words.

INFLECTIONS in a word are structural changes, usually through affixes, to express Number, Tense, Case, Gender, Person, etc.

dog – dogs   goose – geese   hunt – hunted
his – hers

WORD FORMATIONS includes a group of words that have a specific meaning when they appear together.

mother in law   future plan

Now trying to Language Processing implemented of Gujarati Language in two character for one word and maximum two suffixes. There is some basic rule for Gujarati language which is I implemented in my RELAN algorithm.

The Gujarati alphabet consists of 47 letters ordered according to phonetic principles (below each one the standard transliteration is shown followed by its International Phonetic Alphabet equivalent). In below table there are 11 vowels used in Gujarati language.

Gujarati Vowels											
	a	ā	i	ī	u	ū	ṛ	e	ai	o	au

Gujarati has three genders (masculine, neuter rand feminine), two numbers (singular and plural) and three cases (nominative, oblique/vocative and locative) for nouns. The gender of a noun is determined either by its meaning or by its termination. The nouns get inflected on the basis of the word ending, number and case.

The Gujarati adjectives are of two types – declinable and indeclinable. The declinable adjectives have the termination –ũ in neuter absolute. The masculine absolute of these adjectives ends in -o (ો) and the feminine absolute in -ī (ી). For example, the adjective (sārũ - good) takes the form (sārũ), (sāro) and (sārī) when used for a neuter, masculine and feminine object respectively. These adjectives agree with the noun they qualify in gender, number and case. The adjectives that do not end in -ũ in neuter absolute singular are classified as indeclinable and remain unaltered when affixed to a noun.

The Gujarati verbs are inflected based upon a combination of gender, number, person, aspect, tense and mood. There are several postpositions in Gujarati which get bound to the nouns or verbs which they post position. e.g. -nũ (: genitive marker), -mā̃(: in), -e (ે: ergative marker), etc. These postpositions get agglutinated to the nouns or verbs and not merely follow them.

Gender	Singular ()	Plural ()
Masculine	(saar + o) or (sāro)	(saa + i) or (sārā)
Feminine	(saar + i) or (sārī)	(saar + i) or (sārī)
Neuter	(saar + uN) or (sārũ)	(saar + āN) or (sārā̃)

5 RELAN Algorithm

Step 1: Generate an object of obtain the optimal split position for each only two stem and only one suffix Gujarati word in the word list provided for training face data input stream and buffer_reader classes respectively ''data_input_stream', 'buff_read'.

File_writer guj_char= new File_writer("/usr/src/linux-source—2.6.8/kernel/guj_char.c";

Buffer_writer guj_char = new buffer_writer(guj_char);

{ stem1 + suffix1, stem2 + suffix2, stem3 + suffix3, ... , steml + suffixl }

guj_char -> guj_word [2] [N] array   // separating character from text.

f(i) = i * log (freq(stemi)) + (L – i) * log (freq(suffixi))

where   i : Split position (Varies from 1 to L)

L : Length of the Word

STEP 2: Repeat Step 1 until the optimal split positions of all the words remain unchanged.

Loop   { f(i) = i * log (freq(stemi)) + (L – i) * log (freq(suffixi))   }

STEP 3: Generate signatures using the stems and suffixes generated from the training phase.

1St Loop { // To get every character from string [f(i)]

Volume : 5 | Issue : 2 | Feb 2015 | ISSN - 2249-555X

2nd Loop { // To get suffix from the string [f(i)]

} // 2nd Loop

} // 1st Loop

STEP 4: Discard the signatures which contain either only one stem or only one suffix.

Class buff_read closed

boolean stem;

boolean suffix;

write to "guj_char.c ";

if (guj_char=="stem");

write to "guj_char.c "

else if (guj_char=="suffix");

write to "guj_char.c "

close();

6 Applying RELAN Algorithm in booting process

The bootloader is the first software program that runs when a computer starts. It is responsible for loading and transferring control to the Linux kernel.

The grub.conf file is available in /boot/grub/grub.conf. Also the /boot/grub/grub.conf file can also be referenced via the symbolic link file named /etc/grub.conf.

The kernel in the /boot directory is named vmlinuz-2.6.8-1.521, its RAM disk image file is named initrd-2.6.8-1.521.img, and the root partition.

1. Configuring Kernel: Configuring the kernel with the used of terminal as a sudo or root user.

$ make gconfig: - X windows (Gtk) based configuration tool, works best under Gnome Desktop.

2. Compiling Kernel

$ make

$ make modules

$ make modules_install (check user is root or su)

3. Install Kernel

$ make install

It will install three files into /boot directory as well as modification to your kernel grub configuration file.

1. system.map, 2.6.8 2. config-2.6.8 3. vmlinuz-2.6.8

4. Create an initrd image

$ cd /boot

$ mkinitrd -o initrd.img-2.6.8

5. Creating a Custom Kernel

In this step you can create a new system calls in under kernel.h file and this file is available in /usr/src/linux

source—2.6.8/kernel/<file_name>.c

Now generate the simple Gujarati character with the kernel file and linkage with the <file_name>.c with kernel.h

<file_name>.c = guj_char.c

#include <linux/linkage.h>

#include <linux/kernel.h>

#include <asm/uaccess.h>

#define MAX_BUF_SIZE 4

asmlinkage int sys_guj_char(char _ _ usr *buff, int len)

{

char tmp[MAX_BUF_SIZE]; // tmp buffer to copy user's string into

int guj_stem_len; // find how many stems in a string

int guj_suffix_len; // find how many suffixes in a string

a:

if (guj_char=="stem");

write to "guj_char.c "

else if (guj_char=="suffix");

write to "guj_char.c "

close();

File_writer guj_char = new File_writer("/usr/src/linux-source—2.6.8/kernel/guj_char.c";

Buffer_writer guj_char = new buffer_writer(guj_char);

class buff_read closed

// Apply RELAN algorithm

guj_char = { stem1 + suffix1, stem2 + suffix2, stem3 + suffix3, ... , steml + suffixl } ;

guj_char -> guj_word [2] [N] array;

guj_char = i * log (freq(stemi)) + (L – i) * log (freq(suffixi));

boolean suffix;

boolean stem;

printk(KERN_EMERG "Entering guj_char(). The len is %d\n", len);

char guj_char_list;

Volume : 5 | Issue : 2 | Feb 2015 | ISSN - 2249-555X

```
if (len <= 2 || (len > MAX_BUF_SIZE))

{

printk((KERN_EMERG "Entering guj_char() failed: illegal len (%d) !", len); return (-1); } goto (a);

// copy buff from user space into a kernel buffer

if (copy_from_user(tmp, buff, len)

{

printk(KERN_EMERG "Entering guj_char() fail: copy_from_ user() error"); return (-1);

} goto (a)

tmp[len] = '\0';

printk( KERN_EMERG " guj_char() from %s. \n", tmp);

return (0);

close()

}
```

6. Edit the Makefile to compile your new system call

Add guj_char.o to the definition of obj-y, like

obj-y = sched.o fork.o printk.o

cpu.o exit.o resource.o

guj_char.o

7. Add your system call to the unistd kernel header files by editing

/usr/src/linux-source-2.6.8/arch/x86/include/asm/ unistd_32.h. and add the line:

define _ _ NR guj_char 337 // unique system call number

8. Add your system call to the syscalls kernel header files by editing

/usr/src/linux-source-2.6.8/arch/x86/include/asm/syscalls.h. and add the lines

```
/*  kernel/guj_char.c */

asmlinkage int sys_guj_char(char _ _ usr *buff, int len)
```

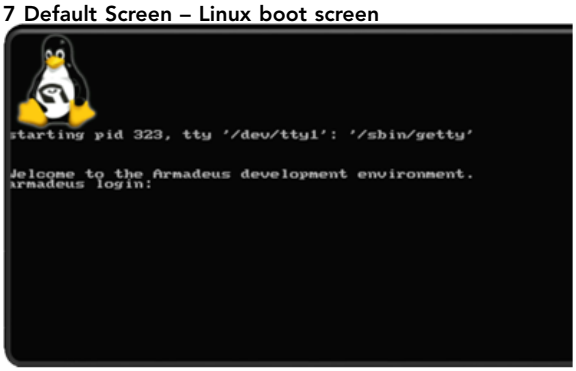
9. Now recompile and load your kernel

Move to the root of the linux source code

```
cd /usr/src/linux-source-2.6.8

make bzImage

make install

# Move to the boot directory

cd /boot

mkinitramfs -o initrd.img-2.6.8.1-cs470p2 2.6.31.9-cs470p2

update-grub

/sbin/reboot
```

10. Now reboot and load new kernel for booting process in your Linux.

7 Default Screen – Linux boot screen



8 Future Booting Screen in Regional Language

9 Conclusion

A revolution is taking place in the way of people, how to access, learn, and interact with information looking booting process in regional language for the Linux based mobile Operating System.

Volume : 5 | Issue : 2 | Feb 2015 | ISSN - 2249-555X

REFERENCE

1. Proposed Booting screen and Architecture in regional language for Linux based mobile devices, By Milan Bhatt, Prashant Dolia [International Journal of Computer Applications, 2013] | 2. Linux Mobile (Architecture) Desktop Software By By Milan Bhatt, Prashant Dolia [Shabd-Brahm] | 3. Memory Management for Many-Core Processors with Software Configurable Locality Policies, By Jin Zhou, Brian Demsky. [International Symposium on Memory Management, 2012] | 4. Characterizing the Memory Management for Improving the Performance of Embedded system used in Wireless Sensor Networks, By Vivek Deshpande, Vijay Wadhai, J B Helonde [IJCA Proceedings on International Conference in Computational Intelligence, 2012]. | 5. Learning to read between the lines using Bayesian Logic Programs. Raghavan, S.; Mooney, R. J.; and Ku, H. ACL 2012. | 6. Proceedings of the 1st Workshop on South and Southeast Asian Natural Language Processing (WSSANLP), pages 51–55, the 23rd International Conference on Computational Linguistics (COLING), Beijing, August 2010. | 7. "Joint Transmitter Power Control and Mobile Cache Management in Wireless Computing", IEEE Transactions on Mobile Computing, Savvas Gitzenis, Member, IEEE, and Nicholas Bambos, Member, IEEE, Vol. 7, No. 4, April 2008. | 8. "An unsupervised Hindi stemmer with heuristic improvements", Amaresh K. Pandey and Tanveer J. Siddiqui, 2nd Workshop on Analytics for Noisy Unstructured Text Data, pp 99-105, 2008. | 9. Prasenjit Majumder, Madar Mitra, Swapan K. Parui, Gobinda Kole, Pabitra Mitra and Kalyankumar Datta, "YASS: Yet another suffix stripper", ACM Transactions on Information Systems, Vol. 25, No. 4, pp 18-38, 2007. | 10. Semantic inference at the lexical-syntactic level Bar-Haim, I. Dagan, I. Greental, and E. Shnarch. 22nd AAAI Conference on Artificial Intelligence, 2007. | 11. Unsupervised models for morpheme segmentation and morphology learning. Association for Computing Machinery Transactions on Speech and Language Processing. Creutz, Mathis, and Krista Lagus. 2007. | 12. "An algorithm for unsupervised learning of morphology", Natural Language Engineering, John Goldsmith, Vol. 12, No. 4, pp 353-371, 2006. |