The Worldwide Multilingual Computing (8): Multilingual Basic Environment – C Language and OS

4 Q – 8

Shoichiro Yamanishi†, Kazutomo Uezono†, Tomoko Kataoka*, Hidejiro Daikokuya†, Toshio Oya†, Tadao Tanaka‡, Kenji Maruyama‡, Yutaka Kataoka* and Hiroyoshi Ohara†

* Centre for Informatics, Waseda University † School of Science and Engineering, Waseda University
‡ Research and Development, Japan Computer Corporation

1. Introduction

The Multilingual Computing Environment was developed by the *Global I/O TM/C System* based on researches of characters, codesets and orthographies [1]. The information brought by the researches of the multilingual computing is the best criterion to define functions of programming languages, device drivers, application softwares as well as libraries and structures of functional modules [Talk 1]. In fact, a lot of problems, insufficiencies and necessities to enhance specifications were found by the information discovered by our researches [Talks 3, 5, 6].

The functions to be enhanced were modified but original calling conventions of the functions were not modified to ensure backward compatibilities. But to realize the environment, a few functions were provided and applications have to call the functions, e.g., for Multi-Locale Model.

Especially, it was found that X server and X font file had a lot of place to improve. In the WASEDA Multilingual Computing Environment (WASEDA MLCE), X protocol itself and font file structures are the same as X11. But by our test, new X server and new font files could increase performance of X server when it draws texts.

By the implementation of the environment, restrictions to process characters were cleared, and larger possibilities of computing came. And WASEDA Multilingual Computing Environment requires minimum changes for applications to be multilingual. But new applications that are designed as multilingual can call more effective additional functions to be multilingual.

2. Programming Language C

First of all, syntax of *Programing Language C* [2, 3] brings an obstacle to multilingual processing. Data type *char* can be interpreted as *signed* or as *unsigned char* by the specification of C. Since the range of an *octet* is from 00/00 to 15/15 and each bit pattern is non-zero value, the type char should be unsigned char. Thus, taking type char as unsigned char by modification of the C compiler itself was selected on the WASEDA MLCE.

WC is also implementation dependent and using WC in a source code does not ensure portability of the code [2, 3, 8, 9]. And generally WC depends on Locale Model except for our system [4]. Thus, our compiler may restrict using WC in a source code by compile switch.

All source codes of the Global I/O TM/C System were carefully programmed to avoid problems caused from the problem above. Therefore, the Global I/O TM/C System can compile and run on other systems.

3. POSIX Libraries

Basically POSIX Locale Model is a bi-lingual model and the problems caused by POSIX Locale Model have been discussed [8, 10]. The *Meta Converter System*, the kernel of the WASEDA MLCE, can absorb system dependencies by no hard-codings that all data are stored in files to be customized [1, 4]. Note that there are some serious problems in Locale related functions still remained.

The *ctype* macros and *wctype* functions were also replaced by calling functions in the Meta Converter System [1, 4]. Also all functions in *string.h* were rewritten.

Since one code point of WC in the WASEDA MLCE is unique among locales, changing a locale does not effect the results of the functions. But relation between a locale and a language is not defined, i.e., a locale does not specify language nor codeset. Thus, collation ordering must not be determined by a locale – specification requires to do so by function strcoll() and others. Collation orderings depend on not only language but also personal requirements. Therefore, collation ordering must be selected from multiple candidates and collation ordering must be done by TMCs [Talk 4]. But the Locale Model does not have such flexibilities. In the WASEDA MLCE, additional functions of the Locale Model can provide the flexibilities.

Note that collation ordering beyond boundary of graphic character sets does not have any rule. The additional functions can set a rule to the situation.

The largest problem in the specification of formatted wide-character input/output functions occurs from mapping between WC codepoints and glyphs. The number of glyph of one WC codepoint is not always one [11, 12, 13, 14, Talk 4]. And when a ligature is presented by two WC codepoints or more, one of such WCs cannot specify a glyph. Therefore, specifying '%c' for one WC codepoint cannot work correctly. Also functions wctomb() and mbtowc() cause the same trouble. By the problems above, POSIX and C specifications clearly need more considerations.

Note that OM in the WASEDA MLCE can return the number of glyphs from one WC codepoint. By this observation of the relation between the number of glyphs and the number of WC codepoints, bitfields of WC were determined in the WASEDA MLCE, i.e., a WC only as 'identifier' does not work.

4. X Window System

The X Waseda Window System (WX) is backward compatible to X11 R5 and R6 in calling conventions of X libraries. And *X protocol* of the WX is fully compatible to X11 R5 and R6.

In order to realize 4 physical directions, an emulator to draw 4 directions is embedded in the OM of WX. But informations in font files in *pcf format* is not enough to draw glyphs vertically. Thus, extra calculation is required to determine the correct location of a glyph when the glyph is drawn vertically. By adding new drawing protocol which draws string vertically can dramatically increase drawing performance of *X server*. To add the new protocol, the pcf format must be added to specify information locating a glyph in a vertical line.

Glyph angle rotation can save memory space allocated by X server. Most of characters, when the characters are drawn in different directions, can specify their shapes by angle rotations. Thus, adding angle rotation ability to the protocol can save memory space spent by X server.

To support improvings above, WX protocol and WX server are now under testing. For the WX protocol, new library X11 can recognize versions of X server and select new protocol or current version. And WX server can accept both new font format and current one.

5. Peripherals

In order to accept any type of codeset, tty drivers should pass 8bit through. But processing input sequence by device drivers is rather difficult because the length of one codepoint is not always one octet. Thus, to solve the problem, the Meta Converter System should be called from such drivers.

Cursor movement handling routines should manage vertical line as well as horizontal line. And such controlling needs determining one 'character' and one 'glyph', so assisting by the Meta Converter System is also required.

6. Kernel of OS

Most of Current *Operating Systems* do not have realtime capabilities which can ensure restricted response time. For the event driven programming for visual and sound managements, the realtime capability is essential. But waiting time in current OSs is quite long by their time-sharing managements. For example, X server has to process incredibly large number of logical events stacked when too many processes are running on a system. Therefore, realtime capabilities for event handling and improvement of re-execution of processes are essential. Those capabilities were designed and those will be integrated into the WASEDA MLCE.

7. The WASEDA MLCE and its Future

As described above, the WASEDA MLCE was designed and implemented. By this total coordination of the system, the WASEDA MLCE is quite effective for daily use with consistencies.

Beyond current results, which are essential for defining and using multilingual environment, a kernel which is more effective to realtime and process management is under development. The new kernel and WASEDA MLCE will provide larger integrity of Multilingual Computing and Multimedia.

References

[1] Kataoka, Y. et al., 1995. Codeset Independent Full Multilingual Operating System: Principles, Model and Optimal Architecture, IPSJ SIG System Software & Operating System, 68-4, pp. 25-32.

[2] ISO/IEC 9899: 1990, Programming language C.

[3] ISO/IEC 9899: 1990/DAM 3, *Draft* Amendment 1:1994 (E), Programming languages – C AMENDMENT 1: C Integrity.

[4] Tanaka, T., et al., Multilingual I/O and Text Manipulation System(4): The Optimal Data Format Converter to/from MB/WC/TMC, Proceedings of the 49th General Meeting of IPSJ, Vol. 3, September 1994, pp 305-306.

[5] Kataoka, Y., et al., Multilingual I/O and Text Manipulation System(1): The Total Design of the Generalized System based on the World's Writing Scripts and Code Sets, Proceedings of the 49th General Meeting of IPSJ, Vol. 3, September 1994, pp 299-300.

[6] Uezono, K. et al., Multilingual I/O and Text Manipulation System (2): The Structure of the Output Method Drawing the World's Writing Scripts beyond ISO 2022, Proceedings of the 49th General Meeting of IPSJ, Vol. 3, September 1994, pp 301-302.

[7] Kataoka, T. et al., Multilingual I/O and Text Manipulation System (3): Extracting the Essential Informations from World's Writing Scripts for Designing TMC and for the Generalizing Text Manipulation, Proceedings of the 49th General Meeting of IPSJ, Vol. 3, September 1994, pp 303-304.

[8] Kataoka, Y. et al., A model for Input and Output of Multilingual text in a windowing environment, ACM Transactions on Information Systems, Vol. 10, No. 4, October 1992, pp 438-451.

[9] ISO/IEC 9945-1: 1990, Information technology – Portable Operating System Interface (POSIX) Part 1: System Application Program Interface (API) [C Language].

[10] Kataoka, Y. et al., A model for Input and Output of Multilingual text in a windowing environment, ACM UIST'91 November 11-13, pp 175-183.

[11] ISO/IEC 2022: 1986, Information processing – 7-bit and 8-bit coded character sets – Code extension techniques.

[12] ISO/IEC 6429: 1992, Information processing – Control functions for coded character sets.

[13] IS 13194:1991, Indian Script Code for Information Interchange – ISCII, Bureau of Indian Standards, India.

[14] TIS 620-2533 (1990), Thai Character Codes for Computers, Thai Industrial Standards Institute, Ministry of Industry, Thailand.