VT-IME: Input Method Editor in FreeBSD vt(4)

Fan Chung
fan@freebsd.org

***Abstract*—This paper introduces the vt-ime framework, specifically designed to enable direct CJK (Chinese, Japanese, and Korean) character input in FreeBSD's virtual terminal, vt(4). Despite vt(4)'s advancements, including Unicode support and the capability to display double-width CJK characters, direct CJK character input capability was still lacking. The vt-ime framework, integrating both backend and frontend components, addresses this gap by providing seamless CJK and non-ASCII character input, thereby enhancing vt(4)'s functionality and usability, especially for users who rely on non-ASCII characters. The backend component is tasked with efficient input processing and character encoding, ensuring both compatibility and performance. Concurrently, the frontend offers an intuitive interface for user interaction. This blend of backend robustness and frontend usability establishes vt-ime as a significant development in FreeBSD, meeting the needs of multilingual environments. The paper also delves into the technical challenges and solutions, and future work of the vt-ime framework.**

***Index Terms*—Input method editor, virtual terminal, CJK characters, FreeBSD**

I. Introduction

The virtual terminal is a text-based interface that allows users to run commands and edit files. In FreeBSD, there are two implementations of the virtual terminal driver : vt(4) [1] and syscons(4) [2]. vt(4) is the newer virtual terminal console driver that replaces syscons(4), and provides several improved features, such as support for Unicode and double-width character for CJK characters. However, vt(4) still lacks support for inputting CJK characters directly.

In this paper, we proposed vt-ime framework, which provides an environment for users to type CJK characters in the virtual terminal vt(4). The vt-ime framework is composed of two parts: the vt-ime backend and the frontend. The backend listens for key events sent from the frontend and uses an input method library, such as *librime* [3], to translate these key events into valid CJK characters. The processed characters are then sent them back to frontend for rendering. The frontend, which runs inside vt(4), is responsible for rendering the virtual terminal and display the CJK characters on the screen.

The source code and documentation of our vt-ime prototype is available at: **https://github.com/Cycatz/GSoC2021**

II. Background

A. Virtual Terminal, CJK and IME

As the software industry has matured and grown in popularity, an increasing number of people are choosing to use Unix-like systems as their primary servers. These systems are used for a wide range of tasks, including general text editing, hosting websites, and even training machine learning models. One of the key tools that is essential for performing these task is a *terminal*, which allows users to access the command line interface and execute commands on the server conveniently.

In FreeBSD, there are two implementations of the virtual terminal driver: vt(4) and syscons(4). vt(4) is the newer virtual terminal console driver that is intended to replace syscons(4) with several added features. These features are enabled by default in the GENERIC configuration for the amd64 and i386 architectures. In particular, vt(4) adds support for UTF-8 encoding and double-width characters, making it possible to display CJK characters.

However, currently vt(4) does not support inputting CJK characters. To type these characters, users must switch to a graphical user interface (GUI) environment and use compatible input method editors. For instance, the X window system [4], the most widely used GUI display framework on Unix-like operating systems, has its own input method protocol called X window input method (XIM) [5]. While using an input method editor in GUI may be convenient for users who use the system as the daily driver, it can be inconvenient and unsuitable for users who use the system as a server. This is because it requires additional programs and libraries to be installed and running in the background, which can consume extra hardware resources and increase costs in terms of both money and time.

The term "CJK" is an acronym for "Chinese, Japanese, and Korean", and the term "CJK character" refers to the Chinese characters or ideographs used in the writing systems of these languages. Standard Chinese is written almost exclusively in Chinese characters, and over 3000 characters are required for general literacy, with up to 40,000 characters for reasonably complete coverage. With such a large number of characters, it is infeasible to map all of them to the keys on a keyboard. Therefore, CJK character users often depend on an additional software called an Input Method Editor (IME) to type CJK characters on a computer.

An Input Method Editor (IME) is a program that allows users to input characters that are not natively available on their keyboard by using sequences of characters. IMEs typically support multiple Input Method Engines, which implement one or more *Input Method*s. An Input Method defines the key sequences that can be used to compose a character.

In Figure 1., the IME *fcitx* [6] is shown along with the input method engine *fcitx-rime* [7]. The input method *Zhuyin*, which is one of the widely used input method in Taiwan, is selected. The text 「你好」 (which means "Hello" in English) is inserted by typing the key sequence "su3cl3" on a QWERTY keyboard. The "preedit string" (labeled (1) in the figure) contains temporary characters that are still being composed and may or may not have been translated into final

CJK characters. The ”candidate selection” area (labeled (2) in the figure) shows the possible results that the input method has produced when translating the keystrokes into a valid character. A user must choose the most suitable candidate from the available candidates.

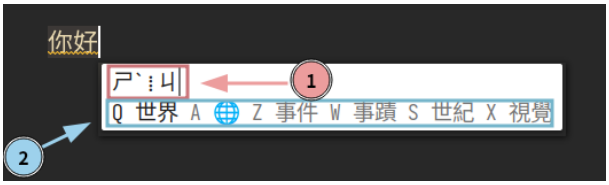

Fig. 1. *Fcitx*, an input method editor, with *fcitx-rime* input method engine and *zhuyin* input method

B. VT-IME

In this paper, we propose *vt-ime*, a system that integrates an Input Method Editor (IME) into the FreeBSD vt(4). vt-ime consists of two parts: a frontend and a backend. The frontend, which runs in the kernel space, is responsible for intercepting key press events and rendering the IME interface in vt(4). The backend, which runs in user space, runs a server that receives sequences of key events from the frontend and translates them into CJK characters by calling APIs from an input method engine.

The steps for typing a CJK character with *VT-IME* are as follows:

1) The user activates the VT-IME mode and presses a sequence of keys.
2) The frontend intercepts the key press events from vt(4) and sends them to the backend.
3) The backend translates the events by invoking the functions in the input method engine library and returns the results to the frontend.
4) The frontend draws the interface for displaying the results, including the preedit text and character/word candidates.
5) The user selects the first candidate using the number key ”1”.
6) The frontend inserts the selected candidate text at the current cursor position.

III. VT-IME

A. frontend

The kernel sources of vt are located in the `src/sys/dev/vt` directory. Here is a list of relevant vt files for our implementation:

- `vt.h`: The main header file, containing struct and function declarations.
- `vt_core.c`: The main source file, containing structure instances and function definitions.
- `vt_buf.c`: A source file defining operations for the vt display buffer.

The vt-ime implementation is based on FreeBSD 13.0-RELEASE, and the functions and data types mentioned in the following text are derived from that version.

To support the input of CJK characters in vt(4), we need to devise a solution for retrieving key events in the terminal. In vt(4), the `vt_process_key` function in `vt_core.c` is responsible for processing key press events. It calls `terminal_input_char`, `terminal_input_special`, or `terminal_input_raw` to send information to the terminal layer, depending on the type of the key.

Therefore, we can leverage key information from the `vt_process_key` function to support the vt-ime mode. To do this, we define the `vt_ime_process_char` function. When the vt-ime mode is enabled, key information is passed to the `vt_process_key` function instead of inputting directly to the terminal. Listing 1. shows the modifications to `vt_processchar` to support this functionality..

```
@@ -990,6 +1012,15 @@ vt_processkey(keyboard_t *kbd, struct
↪  vt_device *vd, int c)
 #if defined(KDB)
        kdb_alt_break(c, &vd->vd_altbrk);
 #endif
-       terminal_input_char(vw->vw_terminal,
-                           KEYCHAR(c));
+
+#if VT_IME
+       if (vt_ime_is_enabled(&vt_ime_default))
+         vt_ime_process_char(vw->vw_terminal,
+                             main_vd,
+                             &vt_ime_default,
+                             KEYCHAR(c));
+       else
+#endif
+         terminal_input_char(vw->vw_terminal,
+                             KEYCHAR(c));
      } else
        terminal_input_raw(vw->vw_terminal, c);

```

Listing 1: Modifications to `vt_processkey`

When a key press event occurs, `vt_ime_process_char`, the core function of vt-ime, perform the following routines in sequence. Each of these routines corresponds to a specific vt-ime function:

1) Send the key to the backend (`vt_ime_send_char`)
2) Display the vt-ime status bar (`vt_ime_draw_status_bar`)
3) Input the characters/words into the terminal if any (`vt_ime_input`)

The process begins by sending key information to the backend. This is achieved by invoking the function `vt_ime_send_char`, which sends the data to the user space via the kernel socket interface. The connection to the backend system is established via a local port that has been previously opened.

The vt-ime status bar is a user interface that displays information related to the IME, including the *preedit string* and a list of word or character candidates. The *preedit string* contains

CJK characters that the user is in the process of composing but has not yet inputted. The candidate area displays a list of available characters or words, which the user can select by pressing the corresponding number key.

In vt-ime, the `vt_ime_draw_status_bar` function is called to draw the vt-ime status bar in vt(4). This function converts the UTF-8 encoded data into `term_char_t` characters using the `vt_ime_convert_utf8_byte` function. `term_char_t` is a data type used to store input/output characters in the terminal layer. The lower bits of `term_char_t` contain the Unicode code point, while the top bits contain other attributes such as colors.

In addition, we need to deal with the display of double-width CJK characters. In vt(4), a double-width CJK character is divided into the left half and the right half, and each half is rendered individually. There is a terminal attribute flag called `TF_CJK_RIGHT` defined in `sys/teken/teken.h`, which can be used to mark a character as the right half of a CJK character by performing a bitwise OR operation with the flag. This allows the terminal to render CJK characters correctly. Listing 2. shows the `vt_ime_draw_status_bar` function.

```

void
vt_ime_draw_status_bar(struct vt_device *vd,
                       char *status)
{
  /* ... */

  ch = FG_WHITE | BG_BLUE;
  len = strlen(status);
  while (len-- > 0) {
    ret = vt_ime_convert_utf8_byte(&utf8_left,

↪  &utf8_partial,
                                   *c++);
    if (ret <= 0)
      continue;
    vb->vb_ime_buffer[blen++] = (ch) |
                                utf8_partial;
    vb->vb_ime_buffer[blen++] = (ch) |
                                utf8_partial |

↪  TFORMAT(TF_CJK_RIGHT);
  }
  /* ... */
}
```

Listing 2: Function `vt_ime_draw_status_bar`

The marco `VTBUF_GET_FIELD` (`vt.h`) is used by vt drivers to retrieve the characters for rendering characters onto the screen. We modify the marco to draw the vt-ime status buffer `vb_ime_buffer` when the requested position is at the first row when the vt-ime mode is enabled. Listing 3. shows the modifications to the `VTBUF_GET_FIELD` marco.

The final step is to input the composed CJK characters by invoking the `vt_ime_input` function. As with the previous steps, the data is converted into a sequence of `term_char_t` characters and inputted using the `terminal_input_char` function.

```
-#define        VTBUF_GET_FIELD(vb, r, c) \
+
+#ifdef VT_IME
+inline term_char_t
+VTBUF_GET_FIELD(const struct vt_buf *vb,
+                int r,
+                int c)
+{
+  if (vt_ime_buf_state && r == 0)
+    return vb->vb_ime_buffer[c];
+  else
+    return ((vb)->vb_rows[((vb)->vb_roffset
↪  + (r)) %
+            VTBUF_MAX_HEIGHT(vb)][(c)]);
+}
+#else
+#define VTBUF_GET_FIELD(vb, r, c) \
    ((vb)->vb_rows[((vb)->vb_roffset + (r)) %
↪  VTBUF_MAX_HEIGHT(vb)][(c)])
+#endif
+
```

Listing 3: Modifications to the `VTBUF_GET_FIELD` marco

B. *backend*

The backend of the vt-ime is responsible for handling key press events received from the frontend, translating them into CJK characters, and sending them back to the frontend. To communicate with the frontend, the backend opens a local port, enters a infinite loop and keeps listening messages from the frontend. In vt-ime, there are five message types *key*, *raw*, *delete*, *output* and *exit*.

The *key* and *raw* message types are used to send visible key and special key information, respectively, to the backend. The *delete* message requests the deletion of a character, while the *output* message requests the current status text, including the preedit string and word candidates. The *exit* message type prompts the backend to terminate.

There are various input method engine libraries available, such as *libime* [8] by fcitx and librime by *Rime* [9]. In the vt-ime, librime was selected due to its user-friendly and highly customizable APIs. Python [10] was chosen as the programming language for backend development due to its native UTF-8 support and high-level interface, which allowed for a greater focus on the core implementation of the backend.

However, as librime is written in C and C++ and its functions and data types are complex, a C wrapper library called *libwrime* was implemented to encapsulate the necessary functions and data types from librime and expose them to Python using *ctypes* [11] . The overall architecture of the backend is depicted in Figure 2.

IV. Result

Figure 3. presents a screenshot of vt-ime within *vim* [12] in vt(4).

The screenshot shows the text 「你好世界」 being inserted in the middle of the terminal. The vt-ime status bar, located at the top of the terminal, displays the preedit string 「ㄋㄧˇㄏㄠˇㄕˋㄐㄧㄝˋ」 and five word candidates: 「我好世界」, 「我好」, 「我」, 「婑」, and 「捰」.

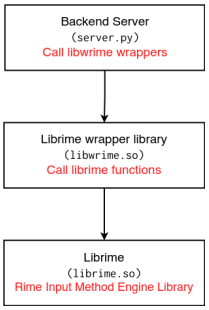

Fig. 2. Architecture of the backend

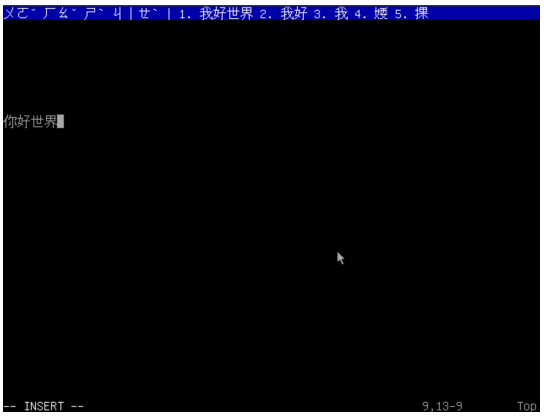

Fig. 3. A screenshot of vt-ime

V. Conclusion & Future Work

In this paper, we propose vt-ime, which offers a solution for typing CJK characters in the FreeBSD vt(4). It consists of a backend and frontend, with the backend translating key events into valid characters using an input method library and the frontend responsible for displaying the characters on the screen. This framework provides a useful environment for users to input CJK characters in the virtual terminal.

However, the current design of vt-ime has some limitations. One of these limitations is the incomplete IME features, as users are currently unable to select candidates using number keys or change the input method. As future work, we plan to improve the integration between vt(4) and vt-ime and enable users to customize vt-ime settings, such as key bindings and input methods, through the sysctl(7) [13] utility.

Additionally, the current communication between the frontend and backend of vt-ime utilizes a socket interface, which may raise security concerns. To address this, we plan to use a character device for communication between the two components and rewrite the backend in C.

References

[1] "vt(4)," a virtual terminal console driver. [Online]. Available: https://www.freebsd.org/cgi/man.cgi?query=vt&sektion=4
[2] "syscons(4)," the console driver. [Online]. Available: https://www.freebsd.org/cgi/man.cgi?query=syscons&sektion=4
[3] "librime," a modular, extensible input method engine. [Online]. Available: https://github.com/rime/librime
[4] "X window system." [Online]. Available: http://www.opengroup.org/tech/desktop/x-window-system/
[5] "X input method protocol." [Online]. Available: https://www.x.org/releases/X11R7.6/doc/libX11/specs/XIM/xim.html
[6] "fcitx," an input method framework with extension support. [Online]. Available: https://fcitx-im.org/wiki/Fcitx_5
[7] "fcitx-rime," rIME support for Fcitx. [Online]. Available: https://github.com/fcitx/fcitx-rime
[8] "libime," a library to support generic input method implementation. [Online]. Available: https://github.com/fcitx/libime
[9] "rime." [Online]. Available: https://rime.im/
[10] "Python." [Online]. Available: https://www.python.org/
[11] "ctypes," a foreign function library for Python. [Online]. Available: https://docs.python.org/3/library/ctypes.html
[12] "Vim." [Online]. Available: https://www.vim.org/
[13] "sysctl(8)." [Online]. Available: https://www.freebsd.org/cgi/man.cgi?query=sysctl&sektion=8