DOI: 10.7763/IPEDR. 2012. V49. 13

Design and Implementation of Embedded GUI Middleware

Hu Sensen

School of Computer Science, Yangtze University, Jingzhou, Hubei, China
foresthss@sina.com

Abstract—The present embedded GUI system supported lib is relatively oversized, complicated and difficult to configure. Owing to the above weaknesses, a new embedded GUI system based on common middleware application to software design was developed. The new embedded GUI middleware adopted POSIX-based multi-thread technology and modularization design; it has been characterized as distinctive structure, real-time performance and sufficient interfaces for further function. This new embedded GUI middleware is successfully applied into satellite compass.

Keywords-middleware GUI embedded system

ARM9 series established as a representative of the low power, high-performance embedded chip, and the cost of TFT LCD screen cut down and touch screen technology progressively developed, embedded system has been widely applied in field of personal entertainment consumption and industrial control. Consequently, graphical user interface (GUI) technology as human-machine interface has gained growing importance. Due to the fact that the present GUI system working under personal desktop environment occupies too much systematic resources and its supporting library as well as linking library also takes up excessive storage space, it is commonly incompatible to embedded system with undersized storage and weak computing capacity.

The current GUI interface of most embedded production is characterized as screen exclusive and single task interactive. In this paper an embedded GUI middleware has been explored to shorten development cycle, the structure as well as the generality of its codes facilitate debugging, maintaining and reengineering.

1. Design Concept of Embedded GUI Middleware

The traditional embedded system development commonly attributes GUI supporting logic to application program, without separating display logic with data processing logic, the lower layer of interface system excessive dependence on hardware causes difficulties in transplant for GUI layer. Ambiguity and high coupling of layer easily cause less-convenient debugging and massive code redundancy.

This paper introduces the middleware concept into exploring which separates GUI from application program as a software layer, making it a separate GUI graphics middleware [1]. The hierarchical relationship in the embedded system is illustrated in Figure 1. Middleware is the common services with standard program interface and protocol located between the platform (hardware and operating system) and application. Program designers take it as an abstract layer in system vertical structure. Through the abstract application, upper application software is served by lower modules, needless to know its specific realization methods. This method simplifies programming module.

Middleware technology shields the distinction of lower hardware platform, facilitating the upper users to gain access to API interface of window control. This technology extracts common interface element and then provides it to the users as a form of an application program interface, through which the application program the users engineered has the same interface characteristic. Developers can flexibly adjust to the target embedded system on demand, which enhances the portability of application program.

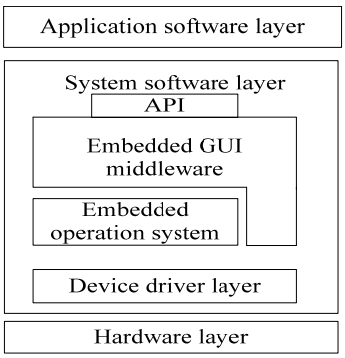

Figure 1.The location of embedded GUI middleware in embedded system

2. Architecture of Embedded GUI Middleware

As an important system software, GUI system should realize a couple of general functions as input and output, graphics operating interface, window management, control and communication mechanism. In embedded system, GUI system requires not only less system resources occupation and portability, high stability and configurability are also expected [2].

Figure 2 presents GUI middleware architecture designed in this paper.

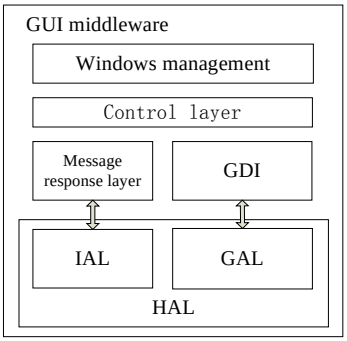

Figure2 GUI middleware architecture

2.1 Hardware Abstract Layer

Hardware abstract layer (HAL) involves input abstract layer (IAL) and graphics abstract layer (GAL).

According to particular purpose and application, embedded system involves different input devices such as keyboard, mouse, touch screen as well as stylus pen. Due to lack of unified standardized interface for present input devices, it is definitely significant to define an input abstract layer which extracts the commonality of input devices and abstracts particular approaches to different input devices. The main interfaces for IAL are given below:

```
typedef struct IAL
{
……
int (* deal_key) (…);  //deal key interface
int (* deal_touch) (…);  //deal touch interface
int (* deal_mouse) (…);  //deal mouse interface
……
}
```

LCD and touch screen are commonly taken as output devices for embedded system. When it comes to Linux-based embedded system, graphics engine is always built on FrameBuffer. A FrameBuffer is a driver interface of display devices on Linux system. It represents the display buffer of some video hardware, and allows application software to access the graphic hardware through a well-defined interface, so that the software doesn't need to know anything about the low-level interface stuff. The graphics abstract layer in this paper builds advanced graphics library based on Framebuffer interface. The main interfaces of Graphics abstract layer are given below:

```
typedef struct GAL
```

```
{
……
int (* gc_context) (…);  //graphics context interface
int (* set_palette) (…);  // palette interface
int (* draw_pixel) (…);  // draw pixel interface
int (* draw_line) (…);  //draw line interface
int (* draw_rectangle) (…);  //draw rectangle interface
……
}
```

2.2 Graphics Device Interface

A graphics device interface (GDI) is above the GAL and transmits basic graphics library functions to GAL. GDI is responsible for tasks such as drawing lines and curves, rendering fonts and handling palettes. The embedded graphics library in this paper avoids floating numbers operation in graphics algorithms. Besides, not merely can users customize the graphics library according to theirs needs to simplify it, they also can add interface in GAL to facilitate the expansion of graphics library.

Meanwhile, the system developed in this paper adopts anti-aliasing algorithm to improve the display quality of graphics. This algorithm smoothes curves and diagonal lines by “blending” the background color with that of the foreground.

2.3 Massage Response Layer

Event-driven mechanism has been used in this embedded GUI middleware. As a contrast to Microsoft’s Windows Message mechanism, event- driven mechanism has less numbers and varieties of messages which are simpler to process. There are more than 20 kinds of messages which are sufficient to meet the needs of embedded systems. Users also can define personal custom message when it necessary. The main messages are:

WIN_CREATE//Windows create message

WIN_SHOW//Windows show message

WIN_PAINT//windows repaint message

WIN_TOUCH//windows process peripheral information of touch screen, receive the touch events from hardware abstract layer and then send out as a message.

WIN_KEY//it processes the messages transmitted from push-button keypad operation to peripherals, push-button can be further divided into button-up and button-down.

WIN_DELETE// message sent when windows deleted.

Synchronous messages and asynchronous messages are two methods of messages sending. Synchronous messages are directly sent to corresponding windows; while asynchronous messages are kept in thread’s message queues for message loop processing [3, 4].

2.4 Control layer

Control layer is a generalized window with message mechanism. it can receive peripheral input events. Control layer possesses some specific functions. Users can repaint on demand. There are some common control layers such as button, label, text box and menu bar defined in the system.

2.5 Windows System

In this paper, windows are set to a unified style, and have support for BMP, JPEG pictures for windows background; windows are of the same size as the screen, which allows users to paint in one particular window. There is no clipping between windows; this system only supports single-task interaction so as to avoid Z order problem. Figure 3 illustrates the event processing.

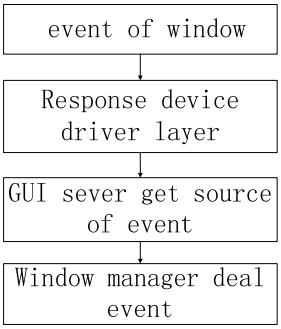

Figure 3 the event processing

Windows system is based on POSIX multithreading technology and uses the multi-process message queue response mode based on mutexes, locks and condition variables. It works as a Client/Server(C/S) mode.

Server is used as Window Manager which is the core of GUI system. It is responsible for desktop management and message distribution and processing in the main message queue. The main functions of sever-side include.

1) Desktop management. Desktop represents window manager at GUI system initialization and then manages the current task window. GUI system supports single-task, so window-manager just manages one current window. All window drawing tasks are accomplished on server-side.

2) Message distribution. Sever is responsible for distribute messages in window manager message queue to corresponding task window correctly and safely.

3) Peripheral input event monitoring.

After starting input devices involving liquid crystal display, touchscreen and keyboard, installing shared resources and initializing device state, the sever turns into monitoring state and begins to message cycling.

Callback mechanism has been used into windows updating. With this mechanism, events can be intercepted before they are transformed to corresponding messages and then transited to particular window. According to return value of callback, events can be passed on along the normal path or be interrupted. There are two mechanisms in callback. Accordingly, users' application program call graphics system, on the other hand, users can return to their application program from graphics system.

Client-side mainly build connection with server-side to transmit event request and then back into message cycling of client side to read corresponding results from its message queue, and subsequently handle those messages.

3. Realization of Embedded Graphics Middleware in Satellite Compass

Satellite compass is navigation equipment rightly for present marine navigation and positioning. It has a promising future as an extended application to GPS positioning and navigation technology.

The satellite compass developed in this paper is precise and complex positioning equipment specific for ocean-going vessel. It provides accurate course and speed of the vessel and reckons the attitude angle. The compass takes S3C2440 based on ARM920T core as CPU and adopts LINUX real-time embedded operation system[5]. The embedded GUI middleware has been applied to human-machine interface, and after compiling, GUI core gets to as small as about 78KB. It works smoothly despite the full load state. Figure 4 shows the GUI interface in the satellite compass working process.

Figure 4 The GUI interface in the satellite compass working process

4. Conclusion

The embedded GUI middleware in this paper has used the characteristics of embedded devices and successfully applied to the satellite compass developing.

The experiment proves the human-machine interface based on the middleware is highly integrated and has excellent real-time performance. Additionally, the middleware possesses good cross-platform performance and efficient hierarchical design which prove scalability for the improvement of the system.

5. Reference

[1] Tammy Noergaard, Embedded Systems Architecture: A Comprehensive Guide for Engineers and Programmers (Embedded Technology) [M]. Newness Publishing, 2005

[2] WeiYongming. MiniGUI [C/OL]. (Oct 1, 2000) http://www.ibm.com/developerworks/cn/linux/embed/minigui/minigui-1/index.html.

[3] YanXiaobing. GUI application program transplantation[M]. Beijing: Publishing house of electronics industry, 2007

[4] Neil Matthew, Richard Stones. Beginning Linux Programming (Third Edition) [M]. Wiley Publishing, Inc, 2007

[5] Karim, Yaghmour. Building Embedded Linux Systems [M]. Nanjing, Southwest University Press, 2007