2024 4th International Symposium on Computer Technology and Information Science (ISCTIS) | 979-8-3503-5456-0/24/$31.00 ©2024 IEEE | DOI: 10.1109/ISCTIS63324.2024.10699253

Optimization Method for Startup Speed of Real-time Process Applications Based on Qt in Embedded Operating System

Yan Wang*
Jiangsu Automation Research Institute, Electronic Equipment Business Unit
Jiangsu, China
wyseume@gmail.com

Abstract—**Aiming at the problems that QT-based real-time process applications in Embedded Operating System operating system are slow to load Qt dynamic library and slow to parse Chinese fonts, which affect the startup speed of the program, this paper proposes a method to optimize the startup speed of real-time process applications. This method implements font pre-rendering and independent romfs loading on the basis of the resource pre-caching method. The experimental results show that the proposed method significantly improves the startup speed of the applications.**

Keywords- Embedded Operating System;Qt; RTP; startup time

I. Introduction

With the development of electronic technology and the continuous improvement of embedded device information, people have higher and higher requirements for embedded graphical user interface. It is a very important work to achieve friendly, rich and high-quality human-computer interaction display on embedded devices. With the increasing complexity of application functions and the continuous improvement of user experience requirements, the startup speed of applications has become a key performance indicator [1-2]. The embedded operating system is compatible with Qt graphical components and extended Qt components, and supports running Qt graphical programs in both kernel mode and user mode. Qt is often used to implement graphical user interfaces based on embedded operating systems because of its powerful cross-platform ability [3].

However, limited by some embedded devices CPU processor processing power is not high, hard disk read and write speed is slow and other factors, when starting RTP applications based on Qt in Embedded Operating System user mode, there is a problem of too long startup time. Specifically, the first RTP program launched in Embedded Operating System is started as a server, when more than one application is running, the subsequent RTP programs connect to the existing server application as clients [4]. Qt applications need to load font files and dynamic libraries into memory when they are started, and also need to calculate each font file to generate font database for display by the application. The loading time of font files and dynamic libraries is long, and the calculation of font libraries, especially the calculation of Chinese fonts, is very time-consuming, which leads to the slow startup speed of Qt applications in Embedded Operating System operating system. In the use of multi-RTP, because the loading time of each RTP is the same, the application with multi-RTP takes longer to load, and the user experience is very bad.

Therefore, in order to solve the above problems, this paper proposes a QT-based multi-RTP application startup speed optimization method in Embedded Operating System operating system. This method uses the font pre-rendering strategy to render the Chinese text fonts used in the program into images, and then uses an independent romfs file system to pre-cache these resources to optimize the application startup speed and improve user satisfaction.

II. Relevant Research Status

In the embedded system environment, due to the relatively limited device resources, such as computing power, memory capacity and storage space, the startup speed optimization of the program has always been the focus and challenge of the research in this field. Therefore, optimizing the startup process is not only the key to improve the user experience, but also the basis for ensuring the system response speed and energy efficiency. In recent years, domestic researchers have conducted extensive and in-depth research on startup speed optimization of embedded systems, and have made remarkable progress.

For the Godson 2K1000B chip, Jiang Guiquan [5] optimized its startup speed through software and hardware analysis, covering PMON, kernel and service initialization. The ESESC tool was used for fine-grained evaluation to identify and reduce problems such as root file system and serial port latency. The measures include cutting the initialization process, optimizing the kernel mount and serial port processing, presetting the lpj value calibration, and implementing the CPU hotplug and STD fast start strategy. At the same time, the memory is recovered and the process is streamlined, which significantly improves the startup efficiency. Aiming at the demand of Chinese display on handheld terminal devices, a fast display method of Chinese vector characters is proposed by adding a proper amount of cache and taking advantage of the hard acceleration ability of GPU [6]. Experimental results show that the display speed of application program interface can be improved by 30-45%, and the page refresh rate can be improved by 40-60%. In order to improve the boot time of the system in the embedded system, Singh [7] et al. optimized the configuration of the Linux-based Android kernel for the embedded system, optimized the impact of changing the software stack on memory consumption, application and function availability, and designed an effective boot loader. Wang Chao [8] et al. propose a method to optimize the boot time of soft-core processor, and rewrites the Bootloader program according to the characteristics of ELF file. In the improved Bootloader flow, the format conversion between SREC and ELF is no longer performed, which greatly reduces the number of memory copies in the boot process and effectively shorts the boot time. Experiments show that the improved Bootloader program is efficient by testing the boot time of two bootloaders under different system main frequency.

To sum up, the research on the startup speed optimization of embedded system programs covers multiple dimensions such as software, hardware and system layers, aiming to effectively improve the response speed and user experience of embedded systems through technical innovation and strategy optimization. As an embedded real-time operating system with independent intellectual property rights, Embedded Operating System, as an embedded real-time operating system with independent intellectual property rights, has been widely used in recent years. For QT-based multi-RTP applications in Embedded Operating System, the research on startup speed optimization is gradually becoming a hot spot in the technology frontier.

III. Optimization Method

Resource pre-caching is a common method to optimize the startup speed of Qt applications. The core idea is to load and cache commonly used resources (images, fonts, style sheets, etc.) in memory before the application starts, so that when the application actually starts, these resources do not need to be loaded again, thus significantly reducing startup time. Aiming at the problems that QT-based multi-RTP applications in Embedded Operating System system are slow to load Qt dynamic library and slow to parse Chinese fonts, this paper proposes a method of font pre-rendering and independent romfs loading based on the resource pre-caching method. The overall architecture design is as follows figure 1.

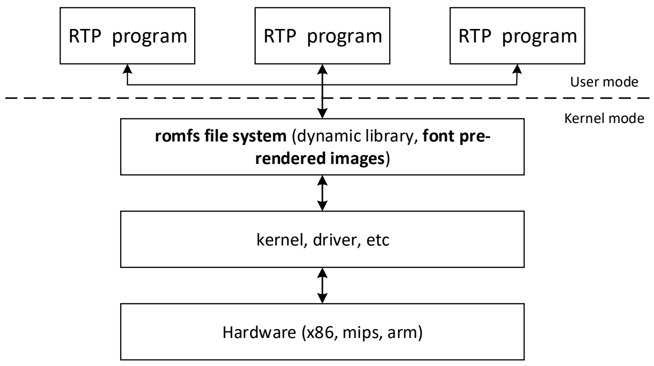

RTP program
RTP program
RTP program
User mode
Kernel mode
romfs file system (dynamic library, font pre-rendered images)
kernel, driver, etc
Hardware (x86, mips, arm)

FIG.1 Overall architecture design drawing

A. *Font pre-rendering*

Through the font processing mechanism provided by Qt, such as QFontDatabase, can deal with all the fonts used by the program, but when the program increases the use of Song typeface, imitation song-dynasty-style typeface or other large number of Chinese characters, in the embedded device with slow hard disk read and write speed, QFontDatabase will still affect the startup time of the program. The comparative analysis of Qt's text drawing performance using raster engine shows that the efficiency of text drawing using the cache after font rendering is many times that of other text drawing methods. This is shown in Figure 2.

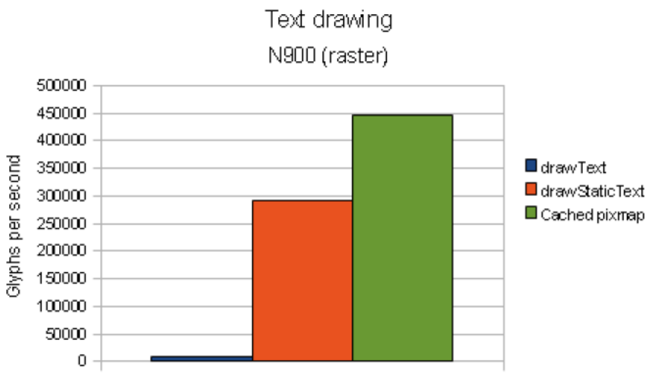


FIG. 2 Comparison of text drawing

Font pre-rendering strategy involves rendering the font of Chinese text used in the program into images and caching these images. In this way, when the application starts, there is no need to parse the Chinese font, and the problem of slow Chinese font computing caused by slow reading and writing speed of the hard disk is reduced, so as to improve the startup speed. Here are the steps

Step 1. Determine the font and text. Determine the font you want to use in your program. Include the font style, size, and type, and list all the Chinese text that may be used in the program.

Step 2. Render the text. For each text that needs to be pre-rendered, the image glyph of each character is encoded as part of the texture using a vector font engine or image editing tool, and finally the rendered character image is saved as an image file (such as PNG, BMP, etc.). These images can be organized in the order in which the characters are encoded, or a single image file can be created that contains all the characters and records its position information for each character. FreeType is the most famous and widely used vector font engine. Its implementation process of loading fonts and rendering them as bitmaps is as follows FIG 3:

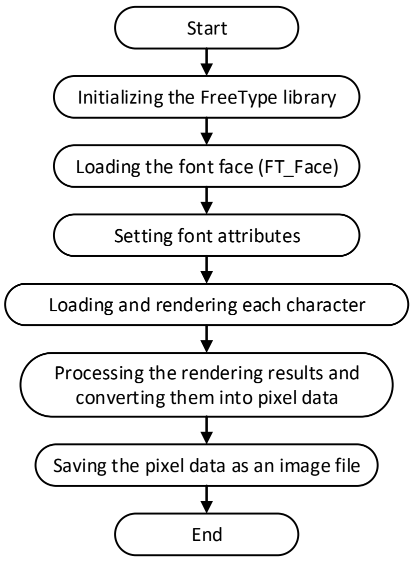

Start
Initializing the FreeType library
Loading the font face (FT_Face)
Setting font attributes
Loading and rendering each character
Processing the rendering results and converting them into pixel data
Saving the pixel data as an image file
End

FIG.3 Text rendering flow chart

Step 3. Manage the cache. In the QT-based RTP program initialization phase, the font image generated in the previous step is loaded into memory and stored as a dictionary or hash table, and the cache management strategy is described in the following section.

Accordingly, we need to modify the way the font is displayed in the application, and use image rendering instead of drawing the text directly where Chinese text is required. The advantage of this method is not only that the font computing process is abolished at the beginning of the program and the startup speed is accelerated, but also that it can be directly reused in the subsequent display process. Combined with the method of using OpenGL to quickly display Chinese characters provided by Kong [6] et al., the burden of text rendering at runtime can be further reduced.

B. Independent Romfs preloaded

For Qt dynamic libraries and resource files that QT-based RTP programs depend on, including the Chinese font rendering pictures described above, this paper proposes a strategy of using an independent romfs file system to pre-cache these resources. Romfs file system is an embedded file system whose file system is read-only and only used to store static files. Embedded Operating System system does not integrate with the romfs filesystem, nor does it provide any romfs filesystem to the user. Since the Embedded Operating System system provides a VxWorks compatible layer, the function is equivalent to VxWorks 6.8[2], so this paper takes the romfs file system of Vxworks 6 as an example to make an independent executable file system for Embedded Operating System. It is installed in RAM along with the Embedded Operating System image at boot time. The steps are as follows:

Step 1. Create a ROMFS memory read-only file system project.

Step 2. Add romfs file system source code, all dynamic library files (*.so) that RTP program depends on, and all kinds of resource files related to RTP program runtime to the project created in step 1;

Step 3. Build and compile the ROMFS project to generate content.romfs image file. The single image file is organized according to the ROMFS file system directory structure, and stores all kinds of dynamic libraries and resource files required by the RTP program;

Step 4. Create the Downloadable Kernel Module (DKM) project;

Step 5. In the project created in step 4, create the romfs_image_prj.s assembly file, create the .section ".rodata "read-only data section, create a global tag named" romfsImage "and a global tag named" romfsImageEnd ". The content.romfs image file generated in step 3 is included in the assembler code using the.incbin assembly directive. The contents of the romfs_image_prj.s file are as follows:

```
.section ".rodata"
    .global   romfsImage
    .global   romfsImageEnd
 romfsImage:
    .incbin   "../../romfs/content.romfs"
 romfsImageEnd:
```

Step 6 Create a blank C language file with .c as the suffix, in this c file,create a function with __attribute__((constructor)) construct attribute.The standard initialization interface usrRomfsConfig of romfs is called inside the function, which is provided by the Embedded Operating System system compatibility layer, so as to realize the automatic initialization of romfs. The code is as follows:

```
__attribute__((constructor)) void deploy()
{
   /* create romfs device */
   usrRomfsConfig();
   cd(FS_ROOT_PATH);
}
```

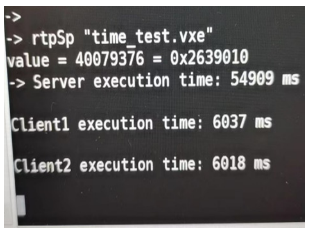
Step 7. Build the DKM project and get the dynamic executable file with the .out suffix

When Embedded Operating System system starts, the .out file can be loaded into ram memory through the loadModule function provided by Embedded Operating System system. The standard romfs initialization interface uses the romfsImage and romfsImageEnd tags created in Step 5 to store the content.romfs image file in memory, using the directory structure created in Step 2. Map out the corresponding file directory and the corresponding files (that is, the various dynamic library files and resource files on which RTP depends). The created romfs file system, in the form of /romfs as the root directory in Embedded Operating System system, provides an in-memory file system for the RTP program.

By setting the LD_LIBRARY_PATH library search path, it points to the directory where *.so is located in the /romfs file directory. When the RTP program is started, the RTP program will find the .so dynamic library and other resource files from the corresponding directory in the memory and start to execute. RTP program starts repeatedly in Embedded Operating System system, dynamic library files and resource files will always be loaded from the memory.

IV. Performance Test

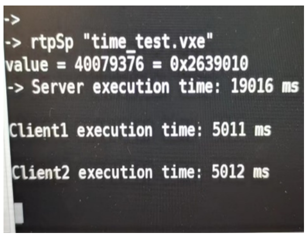
We design three test scenarios for the test of startup time, which are scenario 1 without any optimization measures, scenario 2 with font pre-rendering optimization method, and Scenario 3 with font pre-rendering and independent romfs preloading method. As shown in Figure 4, in these three scenarios, we use three QT-based RTP programs for testing, and the graphical interfaces of the three programs contain 50 to 80 Chinese characters, respectively. When testing, the program is written to schedule the test case, and the three QT-based RTP programs are started in turn in the test case, and the time consumption from the program startup to the interface display is recorded.


```
->
-> rtpSp "time_test.vxe"
value = 40079376 = 0x2639010
-> Server execution time: 54909 ms

Client1 execution time: 6037 ms

Client2 execution time: 6018 ms
```

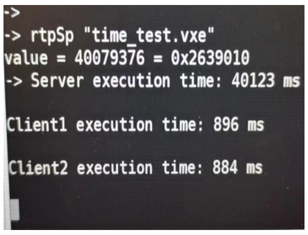
FIG 4.1


```
->
-> rtpSp "time_test.vxe"
value = 40079376 = 0x2639010
-> Server execution time: 19016 ms

Client1 execution time: 5011 ms

Client2 execution time: 5012 ms
```

FIG 4.2


```
->
-> rtpSp "time_test.vxe"
value = 40079376 = 0x2639010
-> Server execution time: 40123 ms

Client1 execution time: 896 ms

Client2 execution time: 884 ms
```

FIG 4.3

FIG.4 Startup time comparison diagram

In the original state, Figure 4.1 shows three program startup times recorded in a startup process. time_test.vxe shown in Figure 4.1 is the program scheduling test case, Server represents the first RTP program, Client1 and Client2 represent the second and third RTP programs respectively.

Following the same operation, multiple startup tests were carried out under scenario 2 and scenario 3 respectively. Figure 4.2 shows the startup time of three programs during a test in Scenario 2. Figure 4.3 shows the time of a test in Scenario 3.

In the three scenarios, we recorded 10 startup data respectively and obtained the following data by calculating the average value, as shown in Table 1. From the data, we can see that the startup speed of the first RTP program is increased by 65% by using the font pre-rendering method, and the startup time of the subsequent RTP program is slightly reduced. When the two methods are implemented at the same time, the startup speed of the first RTP program is increased by 27%, and the startup speed of the subsequent RTP program is increased by over 83%.

Table 1 Test start time comparison table

	First RTP program start time (s)	*Second RTP program start time (s)*	*Third RTP program start time (s)*
Original state	55	6	6
Font prerendering	19	5	5
Font prerender + independent romfs preload	40	<1	<1

In the original state, the first program acts as a server, loading dynamic libraries from hard disk and computing Chinese fonts at startup, which is relatively time-consuming. The next two programs that are launched as clients also need to load the dynamic library from disk, but they can fetch some resources from the server, and the 6-second delay is very bad for the user experience.

By adopting the font pre-rendering method, we remove the Chinese font computing step in the program startup process, and the application can directly load the font rendering image from the hard disk, thus significantly shortening the startup time of the first RTP program. However, since the subsequent RTP programs do not need to compute Chinese fonts, this optimization does not bring significant startup time improvement to the subsequent RTP programs.

Furthermore, we combine font prerendering with standalone romfs preloading. As can be seen from the table, the startup time of the second and subsequent RTP programs is greatly reduced by using these two methods, which is almost negligible, and the average is less than 1s. For the first RTP program, since romfs is also loaded from disk to memory, its startup time is still limited by the disk read and write speed. However, when the two optimizations are implemented simultaneously, the startup time of the first RTP program is still reduced compared to the original state.

V. Conclusions

In this paper, the RTP program startup based on Qt in Embedded Operating System system is analyzed, and the optimization method of font pre-rendering and independent romfs loading is proposed and implemented. The test results show that the optimization effect of the method is remarkable. At the same time, this method is different from the way of coupling romfs and Embedded Operating System system image compilation, which can flexibly configure romfs to form custom .out program without affecting the state of Embedded Operating System system image. The software and hardware structure of embedded system are complex and diverse. These two methods can be flexibly applied in corresponding practical occasions to speed up the program startup and provide better human-computer interaction.

ACKNOWLEDGMENT

This project is supported by the project code 7150206.

REFERENCES

[1] Gui S, Ma L, Luo L, et al. UCaG: an automatic C code generator for AADL based upon Embedded Operating System [C]//2008 International Conference on Advanced Computer Theory and Engineering. IEEE, 2008: 346-350.

[2] CoreTek Systems,2018, Universal Edition Operating System http://www.coretek.com.cn/index.php?m=book&f=read&articleID=11

[3] Amine Brikci N,2024, Qt (software), https://en.wikipedia.org/wiki/Qt_(software)

[4] Qt for Embedded Linux Architecture, 2013, https://het.as.utexas.edu/HET/Software/html/qt-embedded-architecture.html

[5] Jiang Guiquan. Fast startup design and optimization of embedded system based on Loongson 2K1000B [D]. Southeast university, 2022. DOI: 10.27014 /, dc nki. Gdnau. 2020.003751.

[6] XY Kong,XR Kong,(2022),Fast Display Method of Chinese Vector Font Based on GPU Acceleration[C],2022 The 14th International Conference on Artifical Intelligence and Intelligent Information Processing,AIIIP 2022, Qingdao

[7] Singh G, Bipin K, Dhawan R. Optimizing the boot time of Android on embedded system[C]//2011 IEEE 15th International Symposium on Consumer Electronics (ISCE). IEEE, 2011: 503-508.

[8] C Wang,HW Qian,WL Ni. Optimization of Boot Time for Microblaze Soft-core Processor [J]. Microcontrollers & Embedded Systems ,2023, 23(11):17-20.