Acta Universitatis Danubius

COMMUNICATIO

Communication

A Proposed Approach for Web Server Placement Implementation in ARM

Nima Aberomand[1], Ali Eivazy[2], Razieh Takbiri[3]

Abstract: This article developed and implemented an embedded Web server on the S3C44B0X based board by using μClinux operating system as development platform. Two schemes are put forward in this paper: One is an embedded Web server based on boa, the other is an embedded Web server based on HTTP and socket programming. In the first scheme, the main work is to transplant boa in μClinux and to design CGI. The results is realization of dynamic pages and simple control functions. In the second scheme, it is possible to fulfill the GET and POST requests in HTTP and to produce simple dynamic pages.It is also possible to realize the query of history data and some control functions in the request of browser. In this paper, the architecture of embedded Web server and the S3C44B0X development platform is first discussed, the principle and realization of boot loader is also introduced. After that the characteristic of μClinux and network driver is described, including the transplant of μClinux. Then, the principle and mechanism of TCP/IP and HTTP is first discussed and the principle of CGI is also introduced. After that the transplant of boa and the design of CGI and the test of Web server is discussed.

Keywords: Embedded Web Server; μClinux; HTTP; Boa; Socket

[1] Department of Computer Engineering, Shahr-e-Qods Branch, Islamic Azad University, Tehran, Iran, Address: No. 223, Headquarter of Islamic Azad University, South Tehran Branch, ZIP area 11, Azarshahr Street, North Iranshahr Street, Karimkhan-e-Zand Avenue, Tehran, Iran, Correspondng author: nima.aberomand@gmail.com.

[2] Department of Computer Engineering, Shahr-e-Qods Branch, Islamic Azad University, Tehran, Iran, Address: No. 223, Headquarter of Islamic Azad University, South Tehran Branch, ZIP area 11, Azarshahr Street, North Iranshahr Street, Karimkhan-e-Zand Avenue, Tehran, Iran, E-mail: *Ali.eivazy@gmail.com.*

[3] Department of Electronic Technology Engineering, South Tehran Branch, Islamic Azad University, Tehran, Iran, Address: No. 223, Headquarter of Islamic Azad University, South Tehran Branch, ZIP area 11, Azarshahr Street, North Iranshahr Street, Karimkhan-e-Zand Avenue, Tehran, Iran, E-mail: raziyeh.takbiri@gmail.com.

1. Introduction

The purpose of the project is to build an embedded Web server that implements specific functions. It can monitor remote devices. Users can remotely access the Web server through the network to collect data and query historical data. It can also be remotely accessed through various interfaces. The device is controlled. In addition, new control functions can be added as needed without major changes to the web server's framework.

The embedded Web server implemented in this paper is general and can be used in industrial control or smart home systems after improvement, so it has certain practical significance. The system can be used for the collection of physical quantities such as remote temperature and humidity, as well as for the transmission of remote images. Remote control of the device is also possible through the various interfaces of the embedded system.

Due to the popularity of the Internet, various control information can be transmitted quickly and reliably by means of the Internet. Embedded systems have advantages in terms of cost, size, and power consumption. Therefore, combining embedded systems with the Internet is the trend and trend of its development. The fastest growing and most widely used Internet is the WWW service. Web servers and Web browsers provide convenient and stable services. By adding a TCP/IP protocol stack to an embedded device and building a Web server, users can remotely monitor and manage the device through a Web browser. Users can access the embedded web server from any location using standard web browsers (such as IE and Netscape browsers) without having to write any client programs. The embedded Web server can provide rich and colorful information, such as data, text, images, forms, voices, etc. The data can also be updated in real time, and the results of device control can also be immediately fed back.

In industrial applications, it makes sense to use embedded Web servers in areas such as smart devices, instruments and sensors. These devices have an embedded Web server built in, and a dynamic HTML page can be displayed in the user browser, and the system configuration and device parameters can be adjusted in the page. Because of this, the traditional C/S structure control mode is gradually shifting to the B/S structure, which can reduce costs and eliminate the need to develop the client GUI. In summary, the development of embedded Web server has important practical significance and application value. The work done in this paper is mainly reflected in the following aspects:

(1) Construction of embedded Web server software and hardware platform. The hardware platform uses the S3C44B0X development board, and the bootloader is developed as a system boot program on the hardware platform. The software platform uses the µClinux operating system to implement the migration of the µClinux operating system on the development board.

(2) Ported a generic embedded Web server boa, the specific work is to compile and configure boa, as well as the preparation of CGI programs.

(3) Based on the detailed analysis of TCP/IP and HTTP protocols, a specific embedded Web server is designed. The work done is: the preparation of the server main program, the analysis of user requests, the implementation of static and dynamic pages, and the implementation of various control functions according to the user's request.

2. Methodology

Choosing a suitable hardware platform is especially important for embedded systems. This article builds a hardware platform suitable for Web server development with the widely used S3C44B0X processor as the core.

S3C44B0X Microprocessor Overview

At present, ARM chips have occupied about 80% of the market of 32-bit microprocessors. ARM is focused on design, not on chips. The ARM core is known for its combination of high performance, small size, low power consumption, compact code density and multiple supply sources. ARM has become the RISC standard for embedded solutions such as mobile communications, handheld computing, and multimedia digital consumer (Shisheng & Changqing, 2011, pp. 2670-2674). This section introduces Samsung's S3C44B0X chip based on the ARM7TDMI core.

Samsung's S3C44B0X32-bit RISC processor provides a cost-effective and high-performance microcontroller solution for handheld devices and general applications. S3C44B0X integrates ARM7TDMI core on-chip, adopts 0.25um CMOS process, and integrates abundant peripheral function modules based on the basic functions of ARM7TDMI core, which is convenient for low-cost design of embedded application system (Toulson & Wilmshurst, 2017, pp. 3-18).

The S3C44B0X provides the following components: 8KB Cache, optional internal SRAM, LCD controller, 2 UARTs, 4 DMAs, system management (chip select logic, FP/EDO/SDRAM controller), 6 timers with PWM, I/O port, RTC, 8-channel 10-bit

ADC, I2C/I2S bus interface, synchronous SIO interface, clock PLL (Jiyou, Deng, Zhang & Zhou, 2011, pp. 818-824). The characteristics of S3C44B0X can be summarized as follows (Bakos, 2016, pp. 1-47):

✓ Using ARM7TDMI core, I / O voltage 3.3V, core voltage 2.5V.

✓ Built-in phase-locked loop (PLL) with a system frequency up to 66MHz.

✓ 4 working modes for power management to reduce system power consumption.

✓ 8KB System Cache (CACHE), which greatly improves the system speed.

✓ Support 8 MEMORY BANK, maximum external storage space up to 256MB, and support SDRAM.

✓ Built-in color LCD controller.

✓ 2-way asynchronous serial port (UART).

✓ 71 general purpose I/O ports.

✓ 8-channel analog-to-digital converter (ADC).

✓ Real Time Clock (RTC) and Watchdog Circuit (WATCHDOG).

S3C44B0X µClinux operating systemOverview

µClinux is an embedded Linux operating system for industrial control. It is derived from the Linux 2.0/2.4 kernel and follows most of the features of mainstream Linux. It is suitable for microprocessors/microcontrollers without MMU. Supporting CPUs without MMU is a fundamental difference between µClinux and mainstream Linux (Wookey & Tak-Shing, 2002, pp. 52-59).

As an embedded operating system, µClinux has the following characteristics[1]:

✓ Universal Linux API. Those who are familiar with standard Linux will learn to use µClinux very quickly.

✓ The µClinux kernel is small, generally less than 512KB.

✓ µClinux kernel + tool software is generally less than 1M.

✓ µClinux has a complete TCP/IP stack.

✓ µClinux also supports a number of other network protocols.

[1] http://www.µCdot.org/article.pl?sid=03/07/30/0114226&mode=thread.

✓ μClinux supports multiple file systems, including common NFS (network file system), ext2 MS-DOS and FAT16/32, etc.

Compared with standard Linux, since μClinux does not support MMU, multitasking is more troublesome, but most user programs running on μClinux do not need multitasking. In addition, the binary code and source code for the μClinux kernel have been rewritten to tighten and reduce the underlying code. This makes the μClinux kernel very small compared to the standard Linux 2.4 kernel, but it still retains the main advantages of the Linux operating system, such as stability, powerful network features and excellent file system support. In figure 1, S3C44B0X microprocessor architecture block diagram shown.

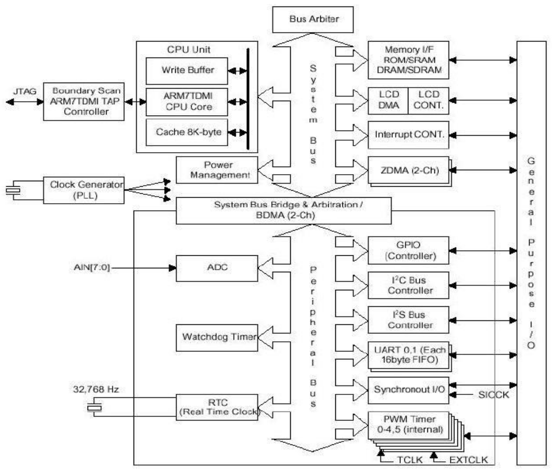


Figure 1. S3C44B0X Microprocessor Architecture Block Diagram

Here are options for processor platforms, library functions, kernels, applications, etc. that can be customized in the configuration. The main configuration options of the system kernel customization are shown in Table I.

Table I. Kernel Customization Options

OPTION	MEANING	SET THE DEFAULT
Vendor/Product	Supplier / Product Selection	Samsung/44B0X
Kernel	Kernel version selection	Linux-2.4.x
Libc Version	Library version selection	μC-libc
System Type	System architecture type configuration	
Generate big endian code	Whether to generate big endian format code	No

Set flash/SDRAM size and base addr	Set the address and size of flash/SDRAM	According to hardware circuit settings
General setup	General configuration	
Networking support	Whether to support the network	√
Networking options	Network configuration options	
Packet socket	Support direct dialogue with network devices	√
TCP/IP networking	Support TCP/IP protocol	√
Block devices	Block device configuration	
RAM disk support	Whether to support RAM disk	√
File systems	File system configuration	
ROM file system support	Whether to support ROMFS	√
Second system support	Whether to support Ext2	√
Network File Systems	Whether to support NFS	√
Character devices	Character device configuration	
Serial devices support	Whether to support serial port	√

The main customization options for the user program of this system are shown in Table 4-2.

Table II. User Program Customization Options

USER PROGRAM	DESCRIPTION	SET THE DEFAULT
Core Applications	Kernel application	
login	a simple login program	√
Sash 、sh	Improved shell program	√
File system Applications	File system application	
Httpd	Another WebServer for embedded applications, simpler than boa, takes up less memory	√
Network Applications	Web application	
boa	WebServer for embedded system applications	√
ftp	FTP client program	√
ping	Network test program ping	√
portmap	TCP port mapper	√

telnet	Remote login client program	√
Miscellaneous Applications	Other applications	
gdbserver	The target system remote debugging program cooperates with the GDB software on the host to remotely debug the program running on the target system.	√
tip	Serial port connection program	√
busybox	A set of tools for embedded applications. Currently implemented many commands, cp, chmod, cat, ln, ls, gzip, ifconfig, mount, etc.	
mount: support NFS mount		√

After the kernel and application are configured, select Save and then compile the kernel and application.

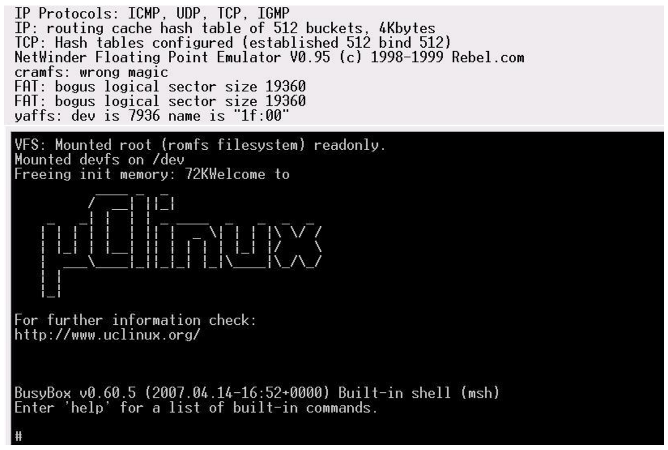
✓ Search for the dependency between μClinux compiled output and source code, and generate dependent files accordingly.

✓ Compile the μC-libc function library to generate libc. a and libm. a function library file.

✓ Compile the user application.

✓ Generate the romfs file system (romfs directory) according to the compiled user program.

✓ Generate a file system image file according to the romfs directory, then compile the kernel and generate a kernel image file. Finally, generate 2 files in the images directory: image. Rom and romfs. Img. At this point, μClinux has been compiled. I have already said that the boot software bootloader burns to the 1 to 4 sectors of Flash, and now the compiled kernel image will be compiled. The rom burns to 5 to 64 sectors (starting address is 0x00010000), and the file system is imaged romfs. Img burns to 65 to 127 sectors (starting address is 0x00100000). Then restart the target board, μClinux will be able to run. The μClinux startup screen is shown in Figure 2.



```
IP Protocols: ICMP, UDP, TCP, IGMP
IP: routing cache hash table of 512 buckets, 4Kbytes
TCP: Hash tables configured (established 512 bind 512)
NetWinder Floating Point Emulator V0.95 (c) 1998-1999 Rebel.com
cramfs: wrong magic
FAT: bogus logical sector size 19360
FAT: bogus logical sector size 19360
yaffs: dev is 7936 name is "1f:00"
```

```
VFS: Mounted root (romfs filesystem) readonly.
Mounted devfs on /dev
Freeing init memory: 72KWelcome to
          ______ _ _
         / _____| (_)
  _   _ | /     | |_ ____  _   _  _   _
 | | | || |     | | |  _ \| | | |\ \/ /
 | |_| || \_____| | | | | | |_| | )  (
 | ._,_| \______)_|_|_| |_|\____|/_/\_\
 | |
 |_|

For further information check:
http://www.uclinux.org/


BusyBox v0.60.5 (2007.04.14-16:52+0000) Built-in shell (msh)
Enter 'help' for a list of built-in commands.

#
```

Figure 2. µClinux Startup Screen

3. Implementation of Embedded Web Server Based on Boa

Boa Overview

Boa is a single-tasking web server that differs from traditional web servers in that boa does not fork new processes for each connection, nor does it fork itself for handling multiple connections. Boa handles all connection requests in a time-sharing manner and only forks the CGI process (this is a separate process). Tests show that boa can process thousands of requests per second, nearly twice as fast as the Apache server (Song, Huaping, Wang; Kong & Zang, 2018, pp. 140-146).

The main goals of boa design are speed and safety. The security of boa means that the web server will not be accessed by unauthenticated users and can encrypt the information. The boa server has an SSL option for security when it is configured. This is not discussed in this article. For details, refer to the related document (Xiuquan, Guoshun, Lei Guo & Yukai, 2014, pp. 276-296).

Boa compilation and configuration in µClinux

First download boa-0.94.14rc21.tar.bz2 from www.boa.org and unzip it under Redhat linux. Then go to the decompression directory and compile boa using the cross-compiler tool arm-elf-gcc. The executable file boa in elf format can be generated. Use the arm-elf-strip -g boa command to remove the included compilation information to reduce the space occupied. The µClinux system uses the romfs file system to require less space than the normal ext2 file system. Therefore, the elf2flt tool is required to convert the generated boa executable file from the elf format to the flat format.

In order to be able to run boa on ARM, you need to set its runtime environment, parameters, etc., and put the final configuration file boa.conf in the appropriate location. The configuration of the web server can be implemented by modifying the configuration file boa.conf. The main configuration options are as follows:

```
Port 80 #Define the port number of the server;

User 0 #Define which users can enter;

Group 0 #Define those user groups can enter;

DocumentRoot /home/web #Define the root directory of the web server;

DirectoryIndex index.html #Define Web Home Name;

MimeTypes /home/web/mime.types #Define the location of the mime.types file;

ScriptAlias /cgi-bin/ /home/web/cgi-bin/ #Define the CGI program directory;
```

The configuration files boa.conf and mime.typesindex.html are stored in the embedded system /home/web directory, then copy the CGI script file Server.cgi to the /home/web/cgi-bin directory, and;

Add the "boa –c /home/web &" command to the /µClinux-dist/romfs/etc/rc file to automatically start the boa web server. Execute commands through the image creation tool;

```
# genromfs –v -f romfs.img -d /µClinux-dist/romfs;
```

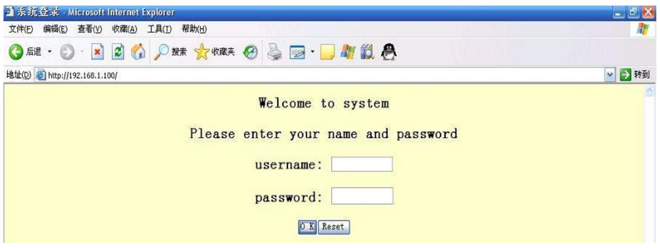
Generate the romdisk image file romfs.img, and download romfs.img to the flash memory through the bootloader. Then restart µClinux, the boa server will automatically run in the background.

Design of CGI Programs in Boa Server

The main job of setting up the boa server is the design of the CGI program and the writing of the page file. This article has written a CGI program based on the working principle of CGI, which can generate corresponding dynamic pages according to the request of the browser, and can realize certain user verification functions and control functions.

Place index.html and related web files in the /µClinux-0408/romfs/home/web directory, and add the “boa –c /home/web &” command to the /µClinux-0408/romfs/etc/rc file. Boa Web server, then compile Server.c with arm-elf-gcc and copy it to the /home/web/cgi-bin directory.

As mentioned above, the files under the entire romfs are packaged with the genromfs tool to regenerate the romfs.img file and burned to the flash. The target board can be restarted to automatically run the boa server. When the user enters the form data on the browser side, the CGI program. The user request is processed and a response message is generated. The target board IP address is set to 192.168.1.100. After typing 192.168.1.100 in IE, the login page shown in Figure 3 is displayed.



Welcome to system

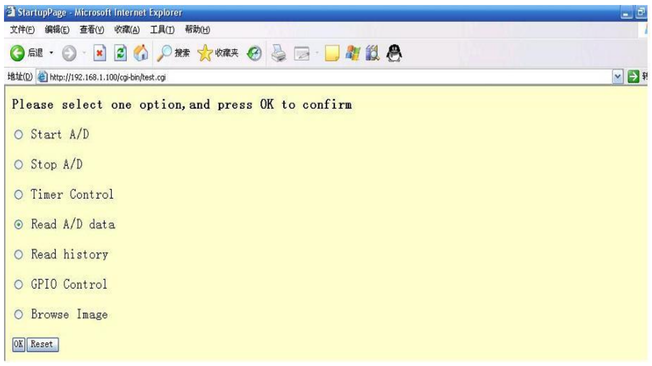
Please enter your name and password

username:

password:

Figure 3. Boa Server Login Page

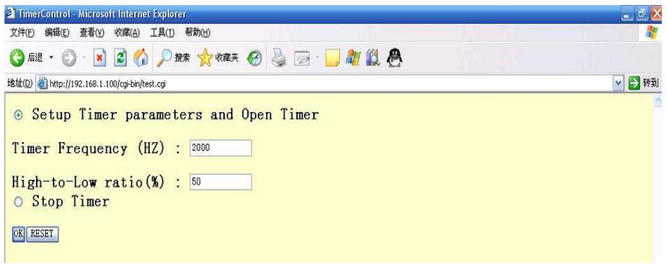
After correctly entering the user name and password, the page shown in Figure 4 will be displayed. This page displays a form. According to the user’s different choices, different requests are generated and sent to the web server. After processing by the CGI program. , then return the results page to the browser. For example, after selecting Timer Control, the page shown in Figure 5 is displayed.


StartupPage - Microsoft Internet Explorer
http://192.168.1.100/cgi-bin/test.cgi
OK
Reset

Please select one option, and press OK to confirm

- Start A/D
- Stop A/D
- Timer Control
- [✓ Read A/D data]
- Read history
- GPIO Control
- Browse Image

Figure 4. Boa Server System Main Page


TimerControl - Microsoft Internet Explorer
文件(F) 编辑(E) 查看(V) 收藏(A) 工具(T) 帮助(H)
后退
搜索
收藏夹
地址(D)
转到
OK
RESET

http://192.168.1.100/cgi-bin/test.cgi

⊙ Setup Timer parameters and Open Timer

Timer Frequency (HZ) : 2000

High-to-Low ratio(%) : 50

○ Stop Timer

Figure 5. Boa Server Control Page

Implementation of a Specific Function Embedded Web Server

The embedded Web server is different from the general server. Considering the limited resources of the system, the design is highly targeted. Some design ideas of the web server in this system are as follows:

✓ The main function of this web server is for the monitoring and configuration of the embedded system, so only the pages related to the system control are implemented.

✓ The web server only processes GET and POST request methods, and does not provide processing for other request methods and CGI.

✓ The web server can only accept one request at the same time, parse the request and return it after processing, and then accept the next request. This is unrealistic for a general-purpose web server, but acceptable for embedded systems.

✓ The web server and the CGI are independent in the general system. In this system, the compact type is considered, and the CGI function is incorporated into the web server.

✓ The system can implement simple user authentication.

✓ The system has some static pages built in, and sends different pages to the browser according to the user's request. When you need to generate a dynamic page, insert the data into a static page.

✓ It realizes the timed task and simple historical data storage function, which can generate data and store it regularly, which can be queried and displayed.

✓ The user can monitor the system through the browser to implement some basic control functions.

4. Conclusion

With the development of embedded systems and network technologies, embedded Web servers are increasingly used in industrial control, smart home and other fields. The embedded Web server function is an inevitable trend in the development of embedded products. This article uses the μClinux operating system as a development platform on the S3C44B0X development board to implement a specific function embedded Web server. The web server can process the GET and POST requests of the HTTP protocol, can form the dynamic data webpage for the user to browse and view, and can also accept the user's request to control the interface of the embedded device, and can also browse the previous historical data. This embedded Web server provides a web-based remote access method for management and control devices, enabling automatic data collection and control in industrial and home areas. Through this research and development, you can explore the embedded Web server development model in ARM+μClinux system. The server program code is written in C language and has strong portability and scalability. In addition, web applications can be easily ported to other embedded operating systems as needed. At the same time, the implementation of the Web server lays a good foundation for the design of more efficient and more complete servers in the future.

References

http://www.μCdot.org/article.pl?sid=03/07/30/0114226&mode=thread.

Jason D. Bakos (2016). Chapter 1: The Linux/ARM embedded platform. *Embedded Systems*, pp 1-47.

Jiyou, Fei; Ran, Deng; Zhao, Zhang & Mo, Zhou (2011). Research on Embedded CNC Device Based on ARM and FPGA. *Procedia Engineering*, Volume 16, pp. 818-824.

Toulson, Rob & Wilmshurst, Tim (2017). Chapter 1: Embedded Systems, Microcontrollers, and ARM. *Fast and Effective Embedded Systems Design* (Second Edition), 2017, pp. 3-18.

Shisheng Jia & Changqing Ma (2011). The Design of the Embedded WEB Server Based on ENC28J60. *Procedia Engineering*, Volume 15, 2011, pp. 2670-2674.

Song, Wei; Liu, Huaping; Wang, Jiajia; Yan Kong & Zang, Weidong (2018). MATHT: A web server for comprehensive transcriptome data analysis. *Journal of Theoretical Biology*, Volume 455, 14 October 2018, pp. 140-146.

Wookey & Tak-Shing (2002). Porting the Linux Kernel to aNew ARM Platform. www.intel.com/pca/developernetwork, Vol. 4, pp. 52-59.

Qiao, Xiuquan; Nan, Guoshun; Tan, Wei; Guo, Lei & Tu, Yukai (2014). CCNxTomcat: An extended web server for Content-Centric Networking," Computer Networks, Volume 75, Part A, pp. 276-296.