Web-based Servo Motor Controller Design with Real-time Micro Embedded Operating System

Ga-Gue Kim*, and Hyung-Seok Lee*

* Embedded S/W Technology Center, ETRI, Daejeon, Korea
(Tel : +82-42-860-1123; E-mail: ggkim@etri.re.kr, hyslee@etri.re.kr)

Abstract: In this paper, we design and implement remote servo motor control system with real-time micro embedded operating system. The system, where controller and camera image grabber are mounted, handles control commands transmitted from a remote PC web browser. A hard real-time servo motor driver running on the real-time micro embedded OS and then a digital control application which confirms precise sampling time intervals is constructed. Frame grabber images transmitted from camera are saved in a image data format to view on remote PC web browser.

Keywords: TCP/IP, embedded system, Qplus, web server

1. INTRODUCTION

Intelligence appliances will become the core of home appliance industries in the digital area. Especially, more and more people are getting interested in controlling intelligence appliances and monitor their status using a web browser in real-time through the Internet. All the above things require a communication protocol called TCP/IP. Even a small appliance should be provided with TCP/IP to communicate through the Internet. Unfortunately, legacy TCP/IP stacks are too heavy to be applied to small appliances that have relatively small memory and low computing power. So it is required to design and implement a TCP/IP stack which is light-weighted enough to be embedded in small appliances[1-5].

In this paper, we attempted to design and implement micro TCP/IP for small intelligence appliances on Qplus real-time operating system[7]. For this goal, we investigated the properties of intelligence appliances, operating systems and requirements needed to connect to the Internet, first. Next, we analyzed the structure and core concept of the BSD based TCP/IP, which are used by QPlus networking module. Based on these investigations, we developed a micro TCP/IP stack that required code size of 38Kbytes on QPlus, including socket API. In comparison with the QPlus network module, where the code size is 140Kbytes, the implemented micro TCP/IP stack is small enough to be applied to the intelligence appliance. In addition, We developed TFTP, DHCP and micro web server, also based on the implemented micro TCP/IP stack.

Qplus is developed by ETRI, this technology consists of reconfigurable embedded Linux kernel, system libraries, graphic window system, and target builder. The target builder is a tool for configuring Qplus. Unlike other which allow only the kernel, this tool will provide the functionality to configure all the components of Qplus like kernel, system libraries, and applications. The system libraries also have been optimized to fit for embedded systems.

2. MICRO TCP/IP

Embedded systems have inherited the programming practices used in larger systems. Network protocols, and TCP/IP in particular, incorporate programming practices used in larger systems. The history of TCP/IP is one of adapting and modifying the original sources written at the University of Califonia at Berkeley to embedded systems. The Berkeley stack is the basis for most of these ports and is the basis of most of the commercial TCP/IP stacks for embedded systems. Of course, real-time and embedded systems face many issues that are unique. A straight port of the Berkeley stack is not the best implementation for the particular needs of an embedded and real-time system. Most vendors have modified the Berkeley code over the years to improve the performance of the stack in embedded systems. Any ports or modifications of the original Berkeley sources should address the following issues[6].

a. Buffer management

The TCP/IP mbuf buffer management should be able to use pre-allocated buffers rather than allocating them from the global heap at run time via malloc.

b. Timers

The times used in the protocols for connection management, timeouts, and retries should be managed by the RTOS. They should not be a separate implementation that will secretly steal bandwidth from the CPU or cause concurrency problems.

c. Latency

If an RTOS is present, it should not add any additional latency. Interrupt-handling interfaces should be fast and deterministic. The RTOS should not add any latency to the interrupt processing required with the physical transmission and reception of a frame. The large amount of context switches and CPU processing required in dealing with a packet increases the importance of using an OS with minimal thread context switch time.

d. Concurrency

All buffering mechanisms should have semaphore protection to allow higher performance potential in real-time systems. The first TCP/IP protocol implementations were on Unix systems and depended on manipulating hardware interrupt levels to eliminate resource contention problems. Semaphore protection should be available to the timers to reduce concurrency problems.

e. Minimized data copying

The TCP/IP implementation should minimize the amount of data copying. The data within each frame can be maintained in the same buffer so it doesn't need to be copied and re-copied by the CPU at each stage of the protocol. The networking chip's DMA places the packets directly in the managed buffer pool where the packet is passed up through the stack by

manipulating pointers and not by copying data. Also, some vendors have extended the mbuf mechanism to allow the data to be shared between mbufs and mblocks where there are STREAMS protocols also present in the system.

f. Link layer multiplexing

Protocol implementation requires a framework with mechanisms for queueing and buffer management. Also, modern protocols require more flexible device driver interfaces and more flexible multiplexing. This is particularly true where serial point-to-point protocols such as PPP are now extended to support IP tunneling and Virtual Private Network (VPN). The original Berkeley implementation isn't sufficiently flexible to meet all of these needs. The better protocol stack implementations use a framework that allows the stack to be extended as new protocols and interfaces are developed. This can be accomplished by extending the basic Berkeley driver interface scheme, or the protocols can be rewritten to use a different framework.

g. CPU bandwidth

Each embedded system application has different requirements for its TCP/IP stack. For example, a TCP/IP stack in most Internet appliances probably would not be considered real time. Also, if the network is used for control and management functions, the hard bandwidth requirements will be fairly low. On the other hand, if the application is streaming video or voice, the faster packet rates would qualify the application as a real-time application.

In this paper, we proceed with modification or addition of the uC/OS-II TCP/IP stack after its porting on Qplus. During our porting process, a few conflicts occurred in system calls part. Therefore, the uC/OS-II system calls was substituted with ones provided on Qplus as follows.

Table 2.1 Substitution of system calls.

system calls	Function	uC/OS-II	Qplus
semaphore	create	OSSemCreate	sema_create
	pending	OSSemPend	sema_wait
	post	OSSemPost	sema_post
time	get OS time	OSTimeGet	clock_get_time
mutual exclusion	CS start	Splx(1)	IC_IRQ_DISABLE
	CS end	Splx(0)	IC_IRQ_ENABLE

The following figure 2.1 shows the overall structure of our TCP/IP stack to be designed and implemented. A frame is received and read from NIC. The frame is generalized in the form of packet to be stored in the buffer by network manager and buffer manager. The packet is transferred to ARP or IP processing module by network daemon. The IP processing module then transfers the packet to ICMP, UDP, and TCP processing module according to the upper protocols. On the other hand, the packet transferred from the upper level is also generalized to be stored in the buffer and then, it is transferred to network device by network manager.

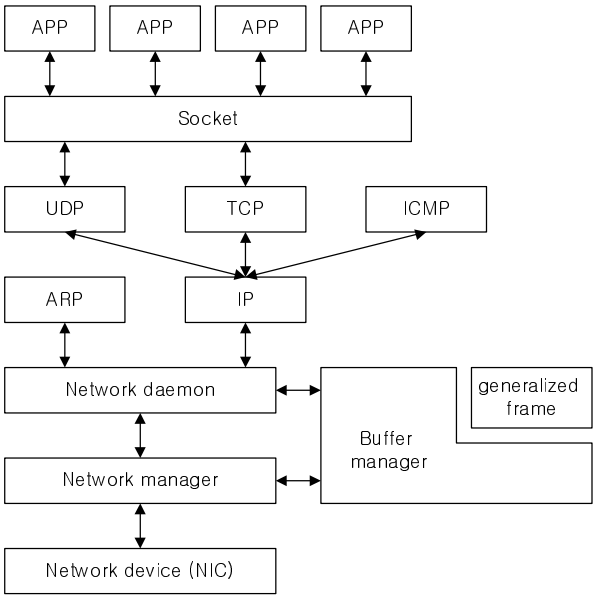

Figure 2.1 Protocol stack structure.

3. Web Server and CGI for micro embedded system

It needs an application level support such as web server to control the embedded system like our servo motor control system remotely. Specially, the remote management mostly has been used in the custom embedded system not having any user interface. However, the existing commercial web servers have excessive many functions and large sizes to be applied to the small-sized embedded systems. A few necessary functions to implement small-sized web server are enough for the remote control embedded system. The functions are summarized as follows

a. File system support

As a web server provides the existing contents via web, the contents should be stored somewhere. In other words, the embedded web server needs a file system for the service contents to be stored.

b. Dynamic contents support

The dynamic contents should be supported in case of adopting the remote monitoring interface in web service, while the most applications need static contents.

c. Form interface support

The form interface has in general the same meaning with CGI(Common Gateway Interface). CGI enables a server application to process data received from web pages. The function which changes embedded system behaviors using the form interface should be support in the embedded web server.

3.1 Web server implementation

Out of the necessary functions mentioned before for the embedded web server, Qplus files system was used for the file system support and CGI also for the dynamic contents and form interface support. Figure 3.1 shows our web server behaviors. A packet data received via allowed port number in web server is transferred to HTTP handler by TCP module. HTTP handler decides whether the packet data is GET method or not by parsing HTTP header, and then whether there exists the requested file or not. It transfers the corresponding file if there exists while it transfers "HTTP 404 NOT FOUND", otherwise.

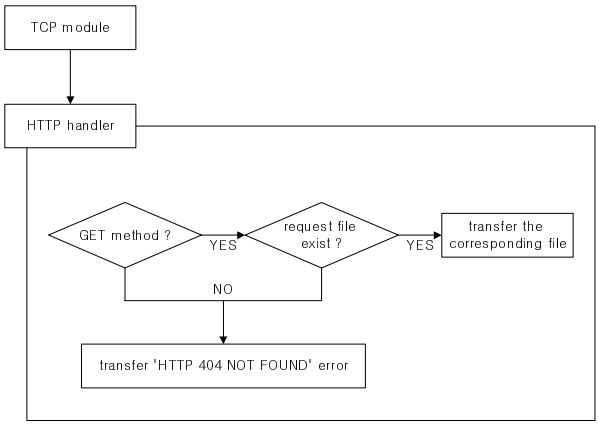

Figure 3.1 Web server behavior.

Figure 3.2 shows pseudo codes for the HTTP handler, including CGI processing code.

```
INT8U HTTP_Handler() {
    parsing_HTTPHeader();
    if(method == GET) {
        if(CGI) {
            if(CGI_task is exists) {
                CGI_task_create(parameter, temp_file_name);
                TCP_Send(temp_file_name);
            }
            else
                TCP_Send("HTTP 404 NOT FOUND");
        }
        else if(HTML) {
            if(requested_file is exists) {
                TCP_Send(requested_file);
            }
            else
                TCP_Send("HTTP 404 NOT FOUND");
        }
    }
    else {
        TCP_Send("HTTP 404 NOT FOUND");
    }
    closesocket();
}

INT8U TASK_WebServer() {
    createsocket();
    bind();
    register_recvTCP(HTTP_Handler);
    listen();

    destroysocket();
}
```
Figure 3.2 Pseudo codes for the HTTP handler.

3.2 CGI supprot

The functions such as conventional CGI modules needs to control an embedded system via web server. It includes system access and dynamic web page generation. The conventional module doesn't fit the embedded system because of excessive many functions and large size.

There are two methods for CGI request using HTTP protocol: GET method and POST method. In the GET method, an argument transferred on request is entered in the request URI, not in the request body of the HTTP header. In the POST method, the argument is conversely entered in the request body of the HTTP header, not in the request URI. Therefore, the request body should be parsed in addition to the URI for the POST method.

The GET method how to manage every request by POST using GET has been widely used for the HTTP request method. We will implement only GET between the two methods. However, it has a weak point that security problems can occurs since the argument is transferred via URI. The original CGI module executes the corresponding CGI file and transfers its results via pipeline. However, our Qplus don't have execution file concept and don't support pipeline. Therefore, we create a task in take place of the execution file and temporary file in take place of the pipeline such as figure 3.3.

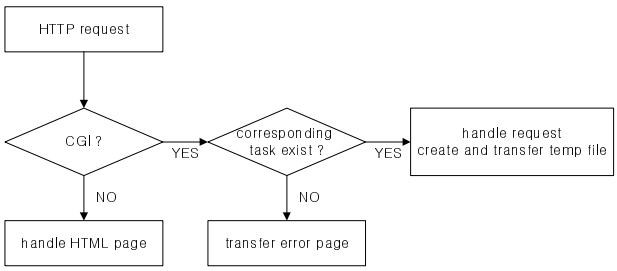

Figure 3.3 CGI behavior for micro embedded system.

The results transferred the CGI task are stored in the temporary file which will be transferred to users. Semaphore will be used for synchronization between temporary file creation and transfer. As soon as CGI request is transferred, the web server executes the corresponding CGI task and waits a semaphore. The CGI task parses and handles the argument transferred from the web server. It also stores its results in the temporary file and posts the semaphore for web server to handle the subsequent process.

```
INT8U HTTP_Handler() {
    parsing_HTTPHeader();
    if(method == GET) {
        if(CGI) {
            if(CGI_task is exists) {
                CGI_task_create(parameter, temp_file_name);
                semawait();
                TCP_Send(temp_file_name);
            }
            else
                TCP_Send("HTTP 404 NOT FOUND");
        }
            …..
}

INT8U CGI_Task(parameter, temp_file_name) {
    results = processing(parameter);
    write_to_tempfile(temp_file_name, results);
    semapost();
}
```
Figure 3.4 Pseudo codes for the CGI task code.

4. WEB SERVER TEST

In this paper, we implemented micro a web server and verified expected behavior. The "index.htm" page which a

user requested displayed on remote web browser. The target board image requested newly was also displayed successfully. Figure 4.1 shows the result when multiple clients have been connected to micro web server loaded on target board. For multiple clients test, we tested requests which were transferred at the same time from three thread browsers on a window system such as figure 4.1, not from separated browsers. We verified that web pages displayed well in the face of three clients connection at the same time. The number of clients connected at the same time can be changed by modifying "socket_num" which expresses the number of sockets.

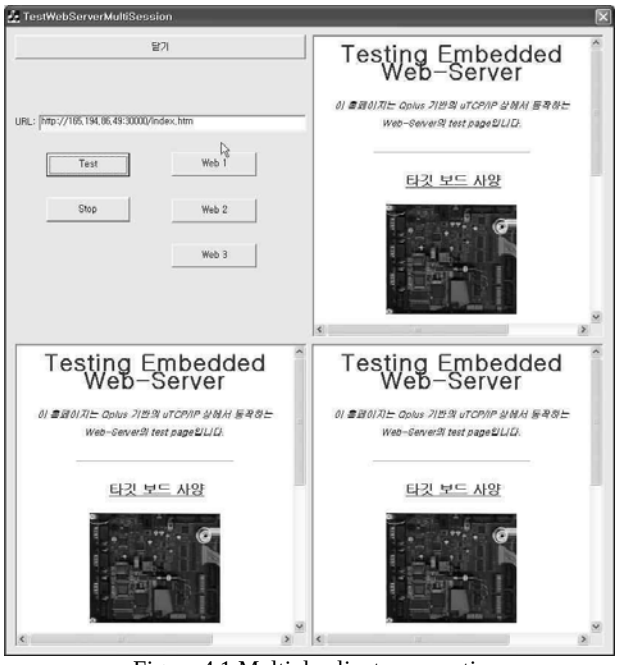

Figure 4.1 Multiple clients connection.

For CGI behavior test, we wrote a simple CGI program to turn on or off LEDs attached on target board via web server. LED control CGI informs a remote user of current LED status via web browser and also provides some buttons to turn on or off LEDs. In this paper, we dealt the number zero port out of 28 GPIOs(General Purpose I/O) on target board based on SA1110 CPU to control LEDs. Table 4.1 shows registers related with SA1110 GPIO control.

Table 4.1 GPIO control registers.

Register address	Name	Permission	Function
0x90040000	GPLR	read only	pin status detection
0x90040004	GPDR	Read / write	input/output direction set
0x90040008	GPSR	write only	output set
0x9004000c	GPCR	write only	output clear

We implement LED control CGI to control and monitor LEDs by accessing above registers. Figure 4.2 shows pseudo codes for LED control CGI using GPIO.

```
GPDR |= GPIO0;

If(CGI_Query is ON)
{
    GPCR |= GPIO0;
}
else if (GCI_Query is OFF)
{
    GPSR |= GPIO0;
}
else
{
    TCP_Send("HTTP 404 NOT FOUND");
}

if((GPLR & GPIO0))
{
    Make_Temp_File(GPIO_LED_is_ON);
}
else
{
    Make_Temp_File(GPIO_LED_is_OFF);
}

TCP_Send(Temp_File);
```

Figure 4.2 Pseudo codes for LED control CGI

5. CONCLUSION

In this paper, we dealt with small size TCP/IP which will be used for general embedded systems as well as small internet appliances. In other words, we ported micro TCP/IP protocol stack on Qplus, real-time micro operating system and also modified TFTP and DHCP. Finally, the embedded web server was developed for controlling internet appliances remotely via internet. TFTP can be behaved as both client and server while DHCP as only client. Both the protocols behave on UDP. The web server behaves on TCP, and deals with basic CGI requests to support remote management and control via internet.

REFERENCES

[1] Jeremy Bentham, *TCP/IP Lean: Web Servers for Embedded Systems 2nd ed.*, CMP books, 2002.

[2] M. Tim Jones, *TCP/IP Application Layer Protocols for Embedded Systems*, Charles River Media, 2002.

[3] Douglas E. Comer and David L. Stevens, *Internetworking With TCP/IP Vol I, 4th ed. : Principles, Protocols, and Architecture*, Prentice Hall, 2000.

[4] Douglas E. Comer and David L. Stevens, *Internetworking With TCP/IP Vol II, 3th ed. : Design, Implementation, and Internals*, Prentice Hall, 2000.

[5] Douglas E. Comer and David L. Stevens, *Internetworking With TCP/IP Vol III : Client-Server Programming and Applications*, Prentice Hall, 2000.

[6] EETIMES NETWORKS, "Embedding TCP/IP", Web page http://www.embedded .com/internet/0001/0001ia1.htm

[7] ETRI Embedded OS Team, "Qplus-P/Target Builder", Web page http://qplus.etri.re.kr/qplus-p