IOSR Journal of Engineering
e-ISSN: 2250-3021, **p**-ISSN: 2278-8719,
Vol. 2, Issue 12 (Dec. 2012), ||V3|| PP 29-33

Implementation of Embedded Web Server with Light weight TCP/IP on Mini 2440

Karthik Bakaraju[1], M Veda Chary[2], Prof M Sudhakar[3]
[1]Department of Electronics & Communication Engineering, CMRCET, Hyderabad, Affiliated to JNTU, Hyderabad.
[2]Assoc. Professor, Dept. of Electronics & Communication Engineering, CMRCET, Hyderabad
[3]Professor, Department of Electronics &Communication Engineering, CMRCET, Hyderabad

***Abstract*— The paper analyses the Light-Weight TCP/IP Stack and gives the detailed processing of every layer first, then selecting the hardware platform as Samsung Mini2440 and the software platform as RT-Thread, porting of LwIP is done based on them. Then a thin web server is designed and stored on embedded device and then the EWS was tested to control a device in this case the DC Motor acting as fan. The result indicated that EWS can remotely monitor and control the devices precisely and perfectly.**

Keywords*— *Embedded Web Servers, Light weight TCP/IP, mini2440, rtthread, embedded web technology.

I. INTRODUCTION

Connecting the embedded device to the internet, implementing perfect Web service on it, and thus realizing a flexible remote monitoring and management through internet browser has already become an inevitable development trend of embedded technology. But due to the limitation of hardware resource and the low-efficiency of general purpose TCP/IP protocol stacks and protocol models, it is quite difficult to implement full TCP/IP protocol into embedded system when accessing to internet. Therefore, we need to port LwIP into the embedded system.

II. LIGHT WEIGHT TCP/IP STACK

LwIP (Light-Weight Internet Protocol) is a small independent implementation of the TCP/IP protocol suite that has been developed by Adam Dunkels at the Computer and Networks Architectures (CNA) lab at the Swedish Institute of Computer Science (SICS). The focus of the LwIP stack is to reduce memory usage and code size, making LwIP suitable for use in small clients with very limited resources.

LwIP features:
1. IP (Internet Protocol) including packet forwarding over multiple network interfaces
2. ICMP (Internet Control Message Protocol) for network maintenance and debugging
3. UDP (User Datagram Protocol) including experimental UDP-lite extensions
4. TCP (Transmission Control Protocol) with congestion control, RTT estimation and fast recovery/fast retransmit.
5. Specialized raw API's for enhanced performance
6. Optional Berkeley-alike socket API
7. DHCP (Dynamic Host Configuration Protocol)
8. PPP (Point-to-Point Protocol)
9. ARP (Address Resolution Protocol) for ethernet

Among the above, important protocols are explained below.

2.1 IP processing

LwIP implements only the most basic functionality of IP. It can send, receive and forward packets, but cannot send or receive fragmented IP packets nor handle packets with IP options. The basic functions of sending and receiving the packets are explained below.

2.2 Receiving packets

For incoming IP packets, processing begins when the ***ip_input()*** function is called by a network device driver. Here, the initial sanity checking of the IP version field and the header length his done, as well as computing and

checking the header checksum. Next, the function checks the destination address with the IP addresses of the network interface to determine if the packet was destined for the host.

2.3 Sending packets

An outgoing packet is handled by the function ***ip_output()***, which uses the function ***ip_route()*** to find the appropriate network interface to transmit the packet on. When the outgoing network interface is determined, the packet is passed to ***ip_output_if()*** which takes the outgoing network interface as an argument. Here, all IPheader fields are filled in and the IP header checksum is computed. The source and destination addresses of the IP packet is passed as an argument to ***ip_output_if()***.

2.4 ICMP processing

ICMP packets received by ***ip_input()*** are handed over to ***icmp_input()***, which decodes the ICMP header and takes the appropriate action. ICMP destination unreachable messages can be sent by transport layer protocols, in particular by UDP, and the function ***icmp_dest_unreach()*** is used for this. The ICMP processing is shown in Fig.1.

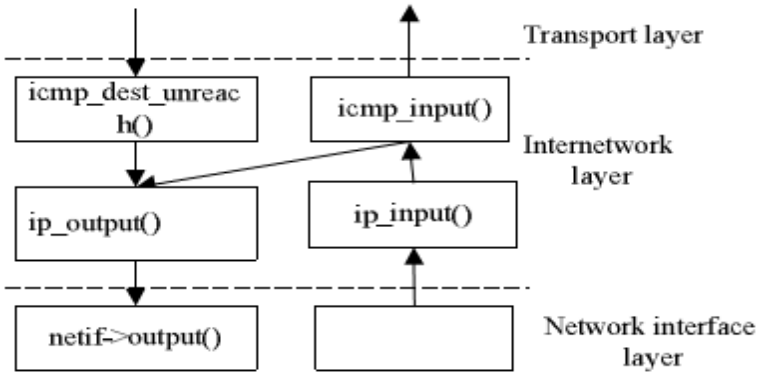

Fig.1 ICMP processing

2.5 UDP processing

UDP is a simple protocol used for de-multiplexing packets between different processes. The state for each UDP session is kept in a PCB structure. The last two arguments recv and recv_arg are used when a datagram is received in the session specified by the PCB. The function pointed to by recv is called when a datagram is received.Due to the simplicity of UDP, the input and output processing is equally simple and follows a fairly straight line (Fig. 2).

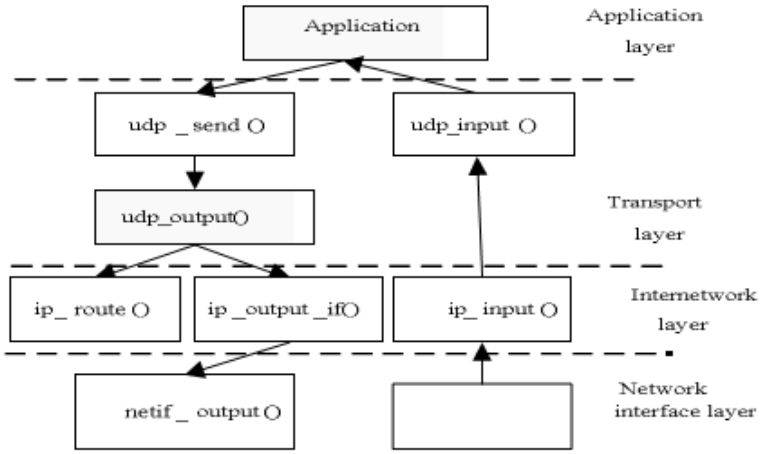

Fig.2 UDP processing

To send data, the application program calls ***udp_send()*** which calls upon***udp_output()***. Here the necessary check-summing is done and UDP header fields are filled. Since the checksum includes the IP source address of the IP packet, the function ***ip_route()*** is in some cases called to find the network interface to which

the packet is to be transmitted. The IP address of this network interface is used as the source IP address of the packet. Finally, the packet is turned over to ***ip_ output_if()*** for transmission.

2.6 TCP processing

TCP is a transport layer protocol that provides a reliable byte stream service to the application layer. The basic TCP processing (Fig.3)

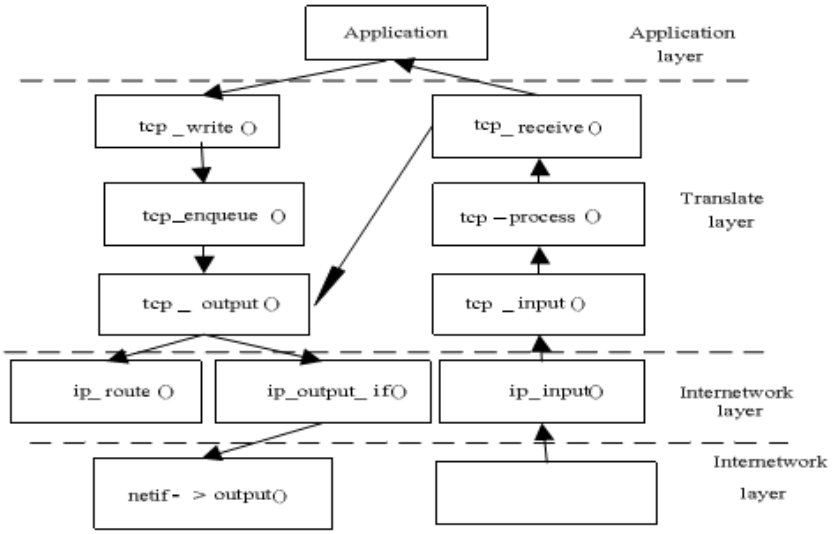

Fig.3 TCP processing

is divided into six functions, when an application wants to send TCP data, ***tcp_write()***is called. The function ***tcp_write()*** passes control to ***tcp_enqueue()*** which will break the data into appropriate sizedTCP segments if necessary and put the segments on the transmission queue for the connection. The function ***tcp_output()*** will then check if it is possible to send the data, input processing begins when ***ip_input()*** after verifying the IP header hands over a TCP segment to ***tcp input()***. In this function the initial sanity checks are done as well as deciding to which TCP connection the segment belongs. The segment is then processed by ***tcp_process()***. The function ***tcp_receive()*** will be called if the connection is in a state to accept data from the network. If so, ***tcp_receive()*** will pass the segment up to an application program. If the segment constitutes an ACK for unacknowledged data, the data is removed from the buffers and its memory is reclaimed. Also, if an ACK for data was received the receiver might be willing to accept more data and therefore ***tcp_output()*** is called.

III. PORTING OF LWIP BASED ON RT-THREAD

In order to make LwIP portable, operating system specific function calls and data structures are not used directly in the code. Instead, when such functions are needed the operating system emulation layer is used. The operating system emulation layer provides a uniform interface to operating system services such as timers, process synchronization, and message passing mechanisms. In principle, when porting LwIP to other operating systems only an implementation of the operating system emulation layer for that particular operating system is needed. The operating system emulation layer provide an interface between the bottom operation system and the LwIP, thus we only need to design some functions in this layer when we port LwIP to a new target operation. Some function about segment, message, time out, new thread and so on. The operating system emulation layer is located in two files ***cc.h*** and ***sys_arch.c***. It provides a common interface between the LwIP code and underlying operating system kernel. The porting to new architectures requires small changes to few header files and a new sys_arch implementation. ***cc.h*** is a basic header that describes the compiler and processor to LwIP. ***sys_arch.c*** provides semaphores and mailboxes to LwIP. For full LwIP functionality, multiple threads support can be implemented in sys_arch file.

IV. REALIZATION OF EWS

The porting of LwIP is the key to realize the EWS. The server based on LwIP we named thin server, it satisfies the embedded device's request. The device information can be uploaded on the web page and appear as data, table and cartoon etc. Embedded web server issue the data to the Internet as the web page so that the remote user can browse the information. From the web page we can know the detailed status of devices and make the relevant control immediately. In application layer, HTTP is the primary protocol; we can monitor the

devices on the spot. Using the HTTP the web server can receive the request data package from clients, read the request message, parse it and send the response to the clients. The state transform of EWS as shown in fig4

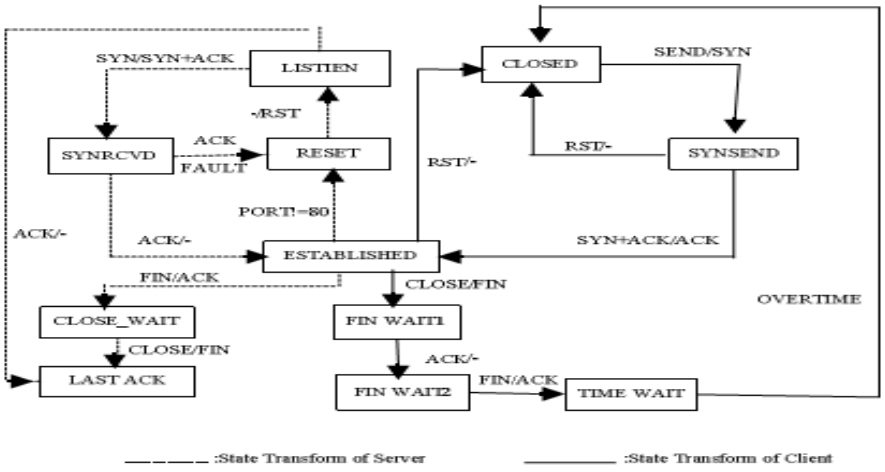

Fig.4 State Transform of EWS

.

The state "RESET" being added into the server, there needn't receive any message to jump to the state of "LISTEN" directly in this state. The port of the server is always open. Once the server finds the "ACK" package is wrong or the server port is not 80 when the connection has been established, the state will jump to "RESET" and send a frame of "RST" to re-establish connection. In order to avoid congestion, once the connection is overtime, it will be closed.

The web server demonstrates the following three features:
1. Accessing files residing on a Memory File System via HTTP GET commands.
2. Obtaining status of the mini2440 board using the HTTP POST command.
3. Controlling the fan connected to the mini2440 board, using the HTTP POST command.

Controlling or monitoring the status of components on the board is done by issuing POST commands to a set of URLs that map to devices. When the web server receives a POST command to a URL that it recognizes, it calls a specific function to do the work that has been requested. The web browser then interprets the data received and updates. There is one main thread in embedded web server which listens on HTTP PORT (PORT 80) for incoming connections. For every incoming connection, a new thread is spawned which processes the request on that connection. The http thread first reads the request, identifies if it is a GET or a POST operation, and then performs the appropriate operation. For a GET request, the thread looks for a specific file in the memory file system. If this file is present, it is returned to the web browser initiating the request. If it is not available, a HTTP 404 error code is sent back to the browser. Using PING command to link EWS in local area network, we can get four response data packages and the time it used is less than 20ms and there is no data package lost. For demonstration of the EWS we have interfaced the mini2440 through a motor driver with a DC Motor which is acting as a fan in this case, and programmed it to regulate its speed of rotation according to the received inputs. When we input the IP address of EWS, we can open the web page through the browser and control the device.

V. CONCLUSION

The implemented EWS is low cost, visualized, platform independent, flexible deployment, excellent remote accessible and can be monitored and controlled flexibly through web pages. This can be used in industrial controlling applications and commercial hi-tech home appliances, on intelligence device, instrument and sensor to realize flexible remote control.

REFERENCES

1). Design and Implementation of the LwIP TCP/IP Stack, Adam D, Swedish Institute of Computer Science, 2001.
2). TCP/IP Lean: Embedded WEB Server, J Benthem, Beijing: China Machine Press, 2003.
3). Ju H T, Choi M J, Hong J W. "EWS based management application interface and integration mechanisms for web based element managements," journal of Network and Systems Management, vol.9, no.1, pp.31-50., 2001.

4). Wei Chen, Porting and Implementation of Light weight TCP/IP Stack; Wireless Communications, Networking and Mobile Computing, 2008. WiCOM '08. 4th International Conference.
5). www.code.google.com/p/rt-thread:: www.rt-thread.org
6). savannah.nongnu.org/projects/lwip/
7). lwip.wikia.com/wiki/porting_for_an_OS
8). http://technet.microsoft.com/en-us/library/bb463206.aspx
9). http://www.a-a-p.org/exec/user-porting.html