International Journal of Computer Information Systems and Industrial Management Applications (IJCISIM)
ISSN: 2150-7988 Vol.3 (2011), pp.009-018
http://www.mirlabs.org/ijcisim

Design and Implementation of Zero-Copy for Linux

Liu Tianhua, Zhu Hongfeng, Liu Jie and Zhou Chuansheng

Shenyang Normal University, China
liutianhua@sina.com, zhfzpku@sina.com, nan127@sohu.com, jasoncs@126.com

Abstract

Zero-Copy has been a hot research topic for a long history, which is an underlying technology to support many applications, including multimedia retrieval, datamining, efficient data transferring, and so on. Zero-Copy means during message transmission, there is no data copy among memory segments on any network node. When a message is sent out, the data packets in user application space go through network interface directly and reach outside of the network; and when receiving a message , the same way is used, meaning the data packets are transmitted into user application space directly. In this paper we present the design and implementation of Zero-Copy technology for the Linux (kernel version 2.6.11) operating system, by modifying its network device driver snull.c and improving on the COW (copy-on-write) technology,. The main method we used is the combination of MMAP and PROC procedures to implement the test program and the test strategies, and finally we successfully simulated the ARP protocol module with the VHDL language.

Index Terms—**Zero-Copy, COW, VHDL, TOE**

I. INTRODUCTION AND RELATED WORK

Nowadays, with the increasing popularity of the Internet, users demand massive bandwidth to transmit increasingly more data and to provide advanced services with high quality. At the same time, the wide deployment of optical fibers enables data transmission at wire speeds. As the transmission speeds are reaching 10 Gbps (OC-192) and heading towards 40 Gbps (OC-768), the bottleneck for high speed transmissions has become the network data processing speed. Special instances for the network processing tasks are the processing functions in the TCP/IP domain. TCP/IP functions are generally referred to as the data processing functions existing in the lower four layers of the TCP/IP model. They are working together to guarantee robust and effective data communications over the Internet. Traditionally, most of the TCP/IP processing functions are performed by software running on general-purpose processors (GPPs). As the network speeds increased drastically, GPPs become burdened with the large amount of TCP/IP processing. Moreover, the processing speeds of some of these functions, especially those computational intensive functions or those functions with high processing overheads, have lagged behind the network speeds. Accordingly, there is an urgent need to identify those performance-critical TCP/IP functions and accelerate them in order to keep pace with the transmission speeds. A challenge in designing the TCP/IP functions is that the demand for advanced services requires the network devices to support a wide range of applications and protocols, however, these applications and protocols are constantly evolving. So the communication bottleneck of the network is moved to the message processing software. In traditional systems, the kernel processes messages, causing many times of data copy and a lot of content transforming and at the end resulting in the high delay between network point-to-points. Therefore, in some developed user level network protocols, the kernel is moved out from the critical message path, this means some or all of protocols are moved into user space from kernel space.

Another trend is to improve the capabilities of NIC. In traditional system architecture, NIC just simply receives data from the host machine and then passes them to network; but an intelligent NIC has programmable processor and memory, so it can take portion or all of message processing tasks from host machine. Therefore, the host machine can focus on its application processing, such as TOE (TCP/IP Offload Engine), etc. Anyway we can summarize these two trends as follows: 1. by using a message processing system to implement message transforming mechanism that walks around the kernel, such as the Zero-Copy technology; 2. by trying the best to use the programmable NIC capabilities and bringing into play of hardware transmission speed, such TOE.

The basic idea of Zero-Copy is: during grouped data transmission from network device to user application space, by reducing the times of data copy and system calls to realize the CPU zero involvement, and totally remove CPU workloads in this processing at the end. The main technologies used to implement Zero-Copy are DMA data transmission technology and memory mapping technology.

Today the implementation of Zero-Copy technology is critical in many systems, such as large scale network intrusion detection systems, network protocol analysis with huge data, high capacity communication systems, different routers, peer-to-peer communication systems, etc The traditional message process has become the bottleneck of the performance of whole system, as it needs two times of data copy at least, one is from network device to operating system memory, and another one is from system memory to user application space; and also it needs the user sending system calls to the operating system. According to Intel reports, the cost of data copy and its related operations has occupied 69% the cost of whole system [1]. Thereby, removing the useless data copy is a critical process

to improve the whole system performance.

Authors in [2] directly used the NIC interfaces and stored them into user space. Pietikainen used an interface of ST (Scheduled Transfer Protocol) protocol and wrote it into intelligent NIC fixed components, to implement OS-bypass [3]. Martin et. al. researched cost problems in NIC [4]. Virtual Interface Architecture (VIA) is a user level industrial standards of network interface [5]. Zero-Copy has important effects for many protocols improvement [6]. Zero-Copy has been implemented in some operating system, such as in Solaris [10] and in embedded operating systems [11]. There are many articles on Zero-Copy, but the characteristics of this article is that we mainly concern about its scalability, adaptability, and configurability in Linux and our implementation can work with software based TCP/IP in transferring simple packets. In addition, we used common socket to connect a user space application in this study and the result is transparent to users.

The paper is organized as follows. In section 2, we present the structural design and analysis of the Zero-Copy model and section 3 describes its implementation, development idea and some key technologies and explanation of source code. Section 4~6 illustrates ARP module function which set an example for the whole design. Finally section 7 presents test program and conclusion.

II. THE STRUCTURAL DESIGN AND ANALYSIS OF ZERO-COPY MODEL

Firstly we analyse the structure of Windows. There are three major models: buffered I/O, direct I/O with MDLs and Direct I/O. Figure 1 illustrates the three modes. In buffered I/O mode the OS allocates a kernel buffer that can handle a request. In the case of a write operation, the OS validates the supplied user space buffer and copies data from the user space buffer to the newly allocated kernel buffer and passes the kernel buffer to the driver. In the case of read, the OS validates the user space buffer and copies data from the newly allocated kernel buffer to the user space buffer. The kernel buffer is accessible to drivers as the AssociatedIrp. SystemBuffer field of an IRP. Drivers read form or write to this buffer to communicate with applications when buffered I/O is in use.Direct I/O is the second I/O method that can be used for data exchanges between applications and a driver. An application-supplied buffer is locked into memory by the OS, so that it will not be swapped out, and a memory descriptor list(MDL) for the locked memory is passed to a driver. An MDL is an opaque buffer through the MDL. The MDL is accessible to drivers through the MdlAddress field of an IRP. The advantage of using direct I/O is that it is faster than buffered I/O since no copying of data to and from user and kernel space is necessary and I/O is per formed directly into a user space buffer.

The third method for I/O is neither buffered nor uses MDLs. Instead the OS passes the virtual address for a user space buffer to the driver. The driver is then responsible for checking the validity of the buffer before it makes use of it. In addition, the user space buffer is only accessible if the current thread context is the same as the application's, otherwise a page fault will occur since the virtual address is valid only while that application's process is active.

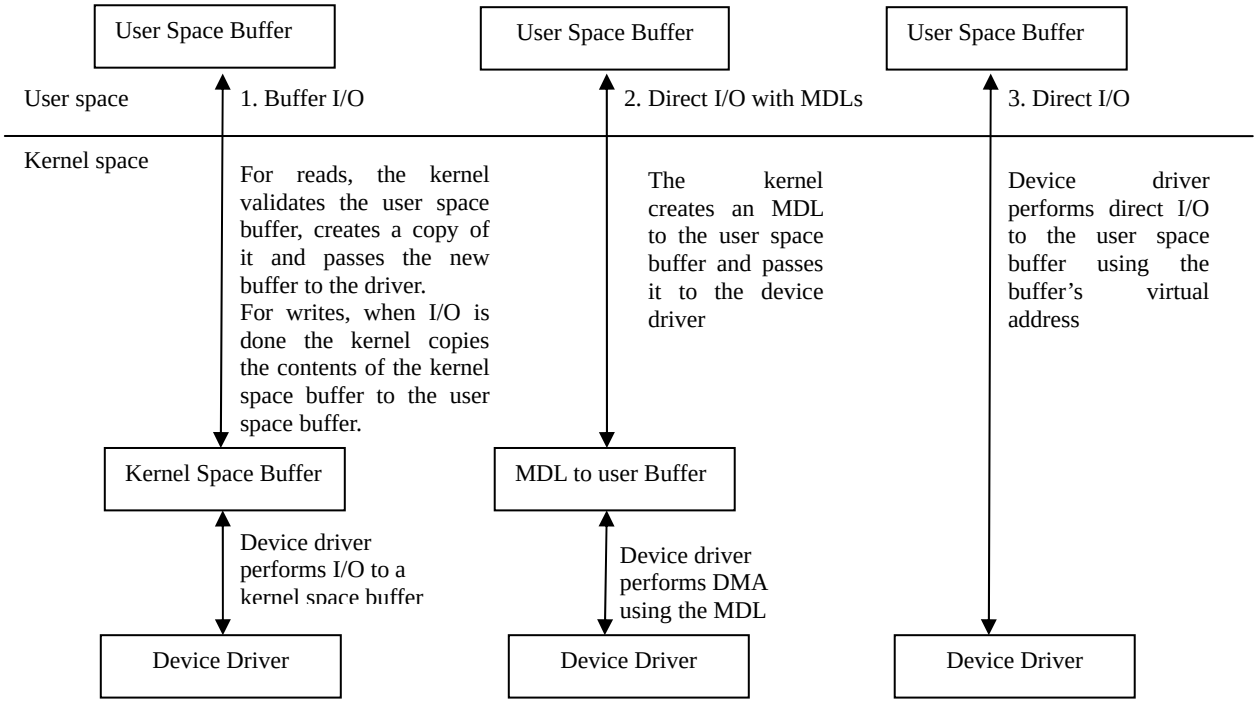

Figure 1. The three ways in which data from kernel to user and user to kernel space is exchanged

In this paper, we only based on network card and by directly modifying the code of network processing in operating system, to implement the Zero-Copy technology. Figure 1 only displays the data receiving procedure. In Linux

system, the normal network data packets receiving procedure is illustrated as below:

1) Network card receives a data packet via to host machine DMA transferred.

2) Network card sends hardware interrupt to host machine.

3) Hardware interrupt: program moves data to receive queue.

4) Software interrupt: program distributes data packets in terms of protocol.

5) User application copies the data packet into user application space by system calls.

6) User application processes the data packet.

Obviously, in traditional model, when application layer wants to obtain message data, it needs to go through two buffers and the normal TCP/IP protocol stack. Inside, the software interrupt is responsible to receive the message from the first of the receiving queue, and then copy them to MSGBuff; at the end application layer reads the message data to user application space by system calls. But in platform with Zero-Copy, it by-passes the protocol stack; when network card is doing DMA operation, it directly transmits the message data to user application space. The broken lines indicated the Zero-Copy implantation in the left of Figure 2. By expanding the broken lines are shown in the right of Figure 2:

Our Zero-Copy implementation contains two steps (here we use data packet receiving as an example):

The first step is to implement DMA data transmission. This step is very closely related to hardware and its device driver. The solution is first to pre-allocate sk_buff (structure) of DMA , and then let the network card receive and store data packets into sk_buff; and finally map the sk_buff into user application space by procedure mmap.

The second step is the address mapping. Linux system provides a technology which can directly map image file and data file into address space of a process. The content of the file can directly map into virtual address space of a process, this is the memory mapping technology. In the kernel, it allocates sequential space as packet pool, and then it can map this space into user application space by the system procedure mmap. In user application space, it can allocate sequential space as packet pool by the procedure sharemem. The link table (physical address mapping table) is stored in kernel, and is used to implement the user application space to physical address mapping, and then the packet address of descriptor received by network card can be read directly from this table.

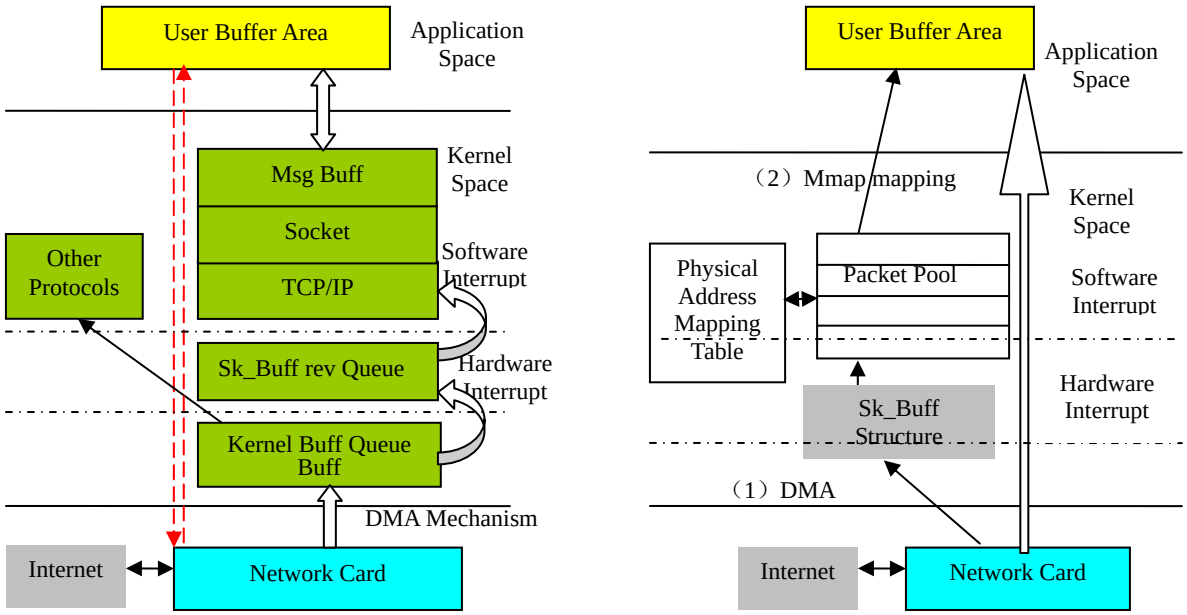

Figure 2. Compare traditional model with zero-copy model

III. THE IMPLEMENTATION OF ZERO-COPY AND ITS KEY THECHNOLOGIES

Now in the TCP/IP network the main communication is interrupt, memory copy and protocol process. Some experiments show that the cost of data copy and its evocable cost (e.g. interrupt, checksum etc.) has occupied 69% of whole cost [1]; in the immediate communication between Linux and FreeBSD, when the data are 64768 bytes and 1460 bytes, the overhead of data copy, context switch and link operate occupy 62%~71% in the total overhead [9]. The latency of kernel to user is 5 times than the latency of user to kernel in the same length just because of misaligned data. It is obvious that for the latency of communication in TCP/IP network, data copy is the main overhead. We show that the COW technology can be improved by removing data copy in the sink node. The COW technology uses already loaded information in the MMU for immediate mapping between user buffer and kernel buffer, so to avoid data copy. All the buffers are stored in the user-space and DMA directly connects the NIC and host.

In detail, we mainly modify the network device driver source code snull.c, by calling a number of procedures (SetPageReserved, learPageReserved, and Page, PROC related procedures) [7][8], to allocate memories in kernel space, and by assign them with some value for user applications accessing them.

A. Memory allocation and its data structure

Here we defined the maximum allocation page as 512, so the size of each buffer is 1500 bytes. In i386 system, the size of a page is 4096 bytes. Then adding variable declaration and struct declaration, by get_free_pages procedure to get the number of PAGES page address in kernel space, data can be transferred into user application space address.

The members of buffer structure are width, length, write pointer and read pointer. Relatively means the buffer size and number of buffers, currently writable pointer and currently readable pointer, as illustrated below:

```
struct MEM_DATA
{ unsigned short width;
unsigned short length;
    unsigned short wi;
    unsigned short ri;
 } *mem_data;
```

The unit of buffer in buffer area, its size is based on a group in real Ethernet, it is 1500 bytes. The size of unsigned int is 4 bytes, the rest space is 1496 bytes, and its structure is illustrated as follow:

```
struct MEM_PACKET
{ unsigned int len;
unsigned char packetp[MEM_WIDTH - 4];
}
```

Two procedures are also needed, the buffer deletion procedure: void del_mem() and the initialization buffer procedure: void init_mem().

During buffer deletion, ,the procedure virt_to_page will change the kernel address to pointer first, and then the procedure ClearPageReserve() sets its flag to no-reserved, after that a loop process is used to set all pre-allocated pages flags to no-reserved and at the end free all pre-allocated memory.

During buffer initialization, the procedure get_free_page gets 2^9 pages buffer for directly access; the procedure of SetPageReserved() can keep pages always in memory so cannot be replaced in order to secure its data; by using a loop process, all buffers are cleaned first and then can be read and written directly.

B. Accessing buffer space

Two procedures are used here: the buffer writing procedure: int put_mem(char* aBuf, unsigned int pack_size) and the buffer reading procedure: int read_procaddr(char* buff, char **start, off_t offset, int count, int *eof, void * data). The buffer writing procedure parameters char* aBuf and unsigned int pack_size contain the content address and the length of content to be written, respectively. The procedure looks up the first buffer with length 0, writes its content and returns its sequential order in buffer space; if lookup reaches to the end of the buffer space, then the lookup goes back to the beginning of buffer space to restart looking up again, unless the whole buffer space has been looked up and the destination buffer location is returned. If the buffer space is full, then rerun 0.

```
int put_mem(char *aBuf, unsigned int pack_size)
{ register int s, i, width, length, mem_i;
char *buf;
struct MEM_PACKET *curr_pack; s = 0;
mem_data =(struct MEM_DATA*)su1_2;
width = mem_data[0].width;
length = mem_data[0].length;
mem_i = mem_data[0].wi;
buf = (void *)((char *)su1_2 +width * mem_i);
for(i=1; i<length; i++)
{
// Lookup all buffer and writing
}
if(i>=length)
s = 0;
return s;
}
```

When reading buffer, procedure _pa() is used to transfer logical address to physical address of the kernel, and then the PROC read procedure is called.

```
int read_procaddr(char *buf, char **start, off_t offset, int
count, int *eof, void {
sprintf(buf, "%u\n", __pa(su1_2)); *eof = 1;
return 9;}
```

In driving procedure snull_cleanup(), the following statements are added: del_mem();

```
remove_proce_entry("nf_addr",NULL);
```

As the procedure snull_cleanup is called when the module is unloaded, here the purpose of doing in this way is to remove our allocated memory and unload the PROC inputs when test is done.

To create PROC inputs in procedure snull_init_module(), we just allocate and initialize the buffer space, and then assign them with certain values.

C. Key technologies and resolutions

Problem 1: Synchronization Problem. The network card driver in the kernel space writes message data to the user application buffer, and the user application processors directly analysis the message data in buffer.

Resolution: In the packet data structure, the flag bit is used to indicate when to read and write. When network card driver writes real message data into this structure, the flag bit is set to READ, indicating the data is readable. After user application processors analyzed and used this data structure, the flag bit is set to WRITE. In the reverse way, the resolution is same. As illustrate in Figure 3, this is the procedure of simplex workload; if in duplex workload, in order to prevent the conflicts and confusions, we can use one flag bit to divide the buffer into 2 partitions.

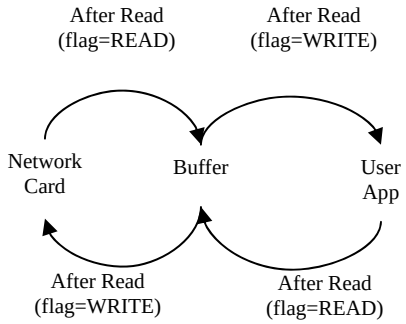

Figure 3. Synchronization Problem

Problem 2: How to make network card and application access the same buffer without any conflicts?

Resolution: We can use the resolution of Problem 1 by adding some protection mechanism to resolve this problem. That is, when multiple connections are working at same time, the mechanism protects the occupied buffer and do not allow any other connections to access it. When initialization sends descriptors, if multiple applications share the same network port, some application may break the descriptor sent by another application. We can use cache to store the active communication node, and store the inactive communication nodes in host memory. When a processor wants to send information through inactive node, network interface will work together with the operating system to activate this node, and moves it to the cache in the network memory.

IV . THE DESIGN AND SIMULATION OF ARP BUFFER MODULE

First we introduce the important characteristics of ARP and the whole ARP data processing.

The key point of ARP running more effectively is that on each host, there is an ARP buffer. Each table filed in buffer stores the mapping record for recent internet address to hardware address. There is a timer for each table field, base on its set clock to remove intact or non-intact table field. For each table field, its normal life cycle is 20 minutes (1200 seconds), the initial time is started at the table creation. During the start of buffer, each table filed lift cycle is set to 0 (means non-useful record). So forth the data writing and query is different from most common data storages. It means before the data writing, it checks each table field life cycle first, if there is some table field which waste all life cycle (means non-useful record), the data writing is allowed, and the life cycle is set to 1200 seconds. If there is not this kind table filed, then the data writing is failure, the status of buffer is overflow (in this design, we don't do special process for overflow, and just discard the record and pass it to upper layer protocol for processing). During data query, it also need checking the table field TTL and just ignore all the records with life cycle equaling to 0.

The structure of table field in buffer is illustrated below; the TTL（time-to-live）means its life cycle. Figure 4 shows the structure of table field in buffer and figure 5 illustrates the whole ARP data processing chart. [13][14].

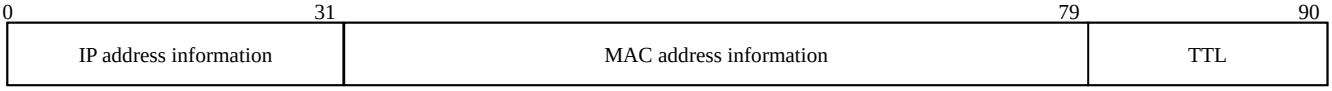

Figure 4. Structure of table field in buffer

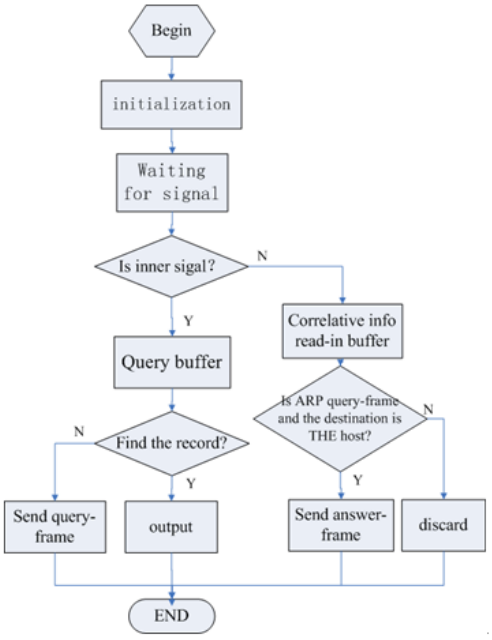

Figure 5. ARP data process chart

A. *Design of state machine*

The state machine of ARP protocol buffer is used to control data writing and query in the buffer， and communication to control module。Based on these functional requirements, we designed 4 states; the state name and their corresponding mean are listed in table 1. [15-17]

Table 1. ARP_BUFFER state machine 4 states and meaning

State	Meanings
BUFFER_MAIN_STATE	Complete data input and state shift
BUFFER_READ	Complete data query
BUFFER_WRITE	Complete data writing
BUFFER_OUTPUT	Complete query result output

Base on Table 1, it generated state shift diagram is described as Figure 6。Start at BUFFER_MAIN_STATE (the CS means segment section signal, signal read is query control signal), when segment selection signal is 1, the query control signal is moving from low to high, the buffer is starting query matching for input data. At this time, the system is on BUFFER_READ state. If the query outputs results, then it turns to BUFFER_OUTPUT state, and output results. After output the results, it turns to BUFFER_MAIN_STATE. If

there is not any query results, then returns “no results” signal, the state turns to BUFFER_MAIN_STATE.

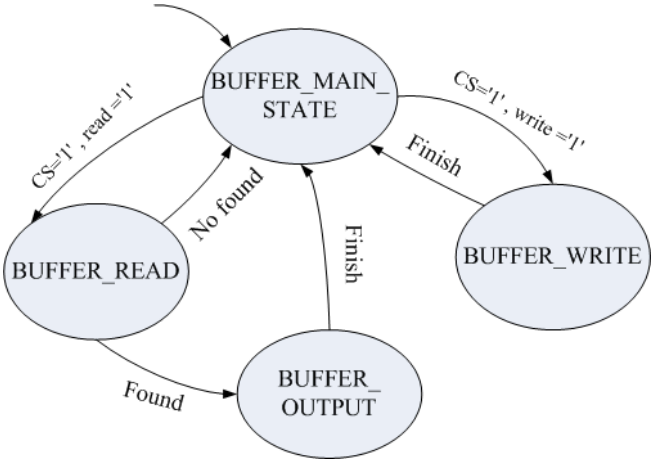

Figure 6. ARP_BUFFER Module State Shift

Signal write is data writing control signal. When segment selection signal is 1, writing signal is moving from low to high, the buffer is start writing operation for input data and the module is turned to state BUFFER_WRITE. After data writing completion, the system returns to BUFFER_MAIN_STATE and waiting for the next input data.

B. Actual Module Design

ARP_BUFFER module ports are illustrated in Figure 7.

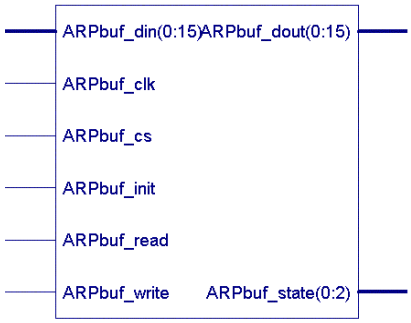

Figure 7. ARP_BUFFER Module Ports Illustration

Input Ports：

*1)*ARPbuf_din(0:15) : Input ports。The data waiting for query and writing is going through this port to enter module. The port width are 16 bits, this is also the pre-setting data bus width.

*2)*ARPbuf_clk : The clock signal of module.

3) ARPbuf_init : Reset signal, high voltage meaningful.

4) ARPbuf_cs : Segment selection signal

5) ARPbuf_read: Query control signal. When segment selection signal is 1, query signal is moving from low to high, buffer start the query matching for input data in the module.

6) ARPbuf_write: Write control signal. When segment selection signal is 1, write signal is moving from low to high, buffer start to writing operation for input data.

Output Ports:

7) ARPbuf_dout(0:15) : Output port. It output query results.

8) ARPbuf_state(0:2) : Buffer state signal, 3 bits, they are mapped to following states :

“000”: Buffer Free State

“001”: Outputting Results

“010”: No Results

“011”: Buffer Overflow (Not processing here)

“111”: Buffer Busy

C. ARP buffer module functional simulation

The main aim of ARP buffer function is to validate query and write in about the buffer. The experiment process: the simulation results are showing in figure 8. First the system begins reset at 100ns and this time the ARPbuf_state is “000”(means buffer idle, wait for data to be input). Here the module keeps itself in BUFFER_MAIN_STATE. Then we write a address information which including IP and MAC at 500ns, just like figure 8, the ARPbuf_cs is 1 and ARPbuf_write is from bottom to top. Because we beforehand set bandwidth is 16bits, both the 32bits of IP address and the 48bits MAC address need 5 clock cycle altogether. We can see when the data input the buffer state is “111” (buffer busy). After a while the module shifts to BUFFER_WRITE state, and then finishes writing address of IP and MAC. In order to validate that, we query the IP address from buffer at1800ns, when the ARPbuf_cs is 1 and ARPbuf_read is from top to bottom, the query data begin to input. Here we can see after IP address has been input a cycle, the state of buffer from “111” shift to “001”, which means the buffer is outputting the result. At the same time the port of ARPbuf_dout output 48bits data which are the MAC address that has been input before. The test result indicates that the function is accord with design original intention.

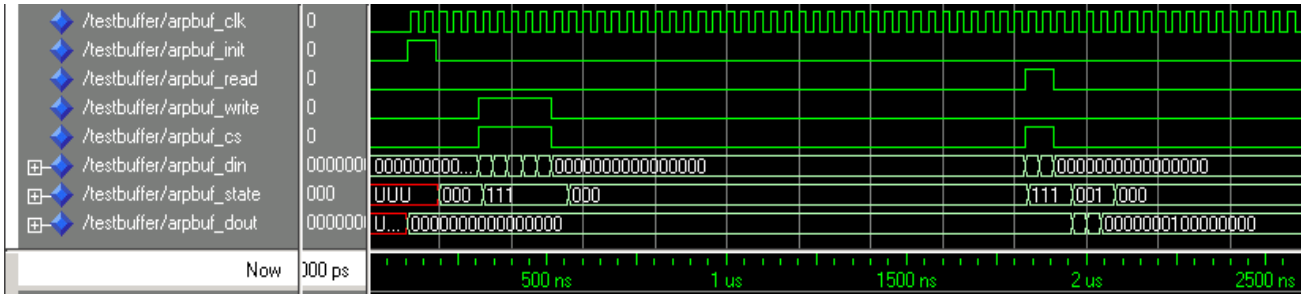

Figure 8. ARP buffer module simulation results

V. THE DESIGN AND SIMULATION OF ARP CONTROL MODULE

Now we design ARP_control module in the base of the ARP_buff module. The basic functions as below:

(1) When IP send a ARP query, ARP control module query buffer; If ARP control module found a record, then sent the record to export; on the contrary, it sent ARP query to network.

(2) When a ARP query message is sent form network, we write the send node address to buffer. If receive node is local address, we send responding message, otherwise discard.

(3) If local node receive a ARP respond from another node, host will keep the address information to buffer, then discard.

A. Design of state machine

According to the basic function of ARP control module, we design 6 states and each of them illuminates as table 2.

Table 2 estates and its meanings

State	Meanings
ARP_CTRL_MAIN_STATE	data import and state shift
ARP_CTRL_REQUEST_INSIDE	query from buffer
ARP_CTRL_OUTPUT_INSIDE	export query result
ARP_CTRL_WAIT	wait when query buffer
ARP_REQUEST_OR_ANSWER	process request and answer message
ARP_SEND_FRAME	sent out data frame

According to table 2, we create state machine as figure 6: ARPctrl_in_source mark data source, “10”stand for query request of MAC from IP. “01”stand for ARP request or respond message from outside. ARP_CTRL_MAIN_STATE will based on these signals to decide which state to shift.

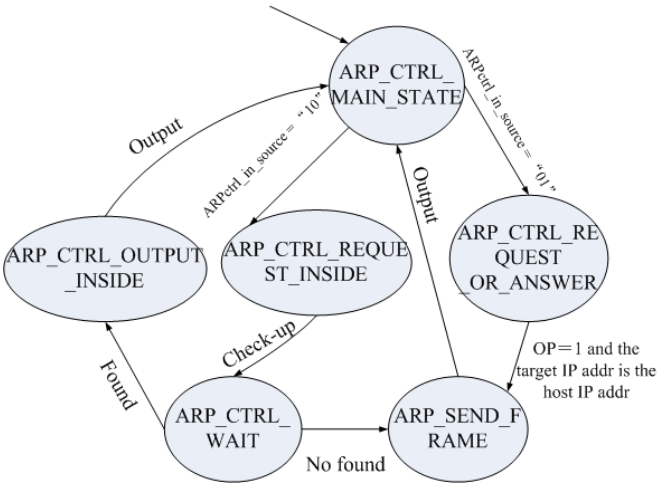

Figure 9. ARP control state shift

B. Actual Module Design

The ports of ARP control module as figure 9 show:

Figure 10. ARP control module ports

PORTS illuminates as below:

Input ports:

1) ARPctrl_clk : clock signal.

2) ARPctrl_reset : reset signal.

3) ARPctrl_buf_in(0:15) : port which takes data from buffer.

4) ARPctrl_buf_state(0:2) : buffer state.

5) ARPctrl_din(0:15) : port which takes data from outside.

6) ARPctrl_in_source(0:1) : identify the data origin of ARPctrl_din (0:15), “10”stand for IP query, “01” stand for network query; “00” stand for no data input.

Output ports:

7) ARPctrl_buf_out(0:15) : port which sends out data to buffer.

8) ARPctrl_dout_to_ip(0:15) : port which sends out query result to IP.

9) ARPctrl_out_ip_en : port enable signal of ARPctrl_dout_to_ip(0:15), high level electricity output is in effect.

10) ARPctrl_send_frame(0:15) : port which sends out data frame.

11) ARPctrl_send_en : port enable signal of ARPctrl_send_frame (0:15), high level electricity output is in effect.

12) ARPbuf_state(0:2) : buffer estate signal，each of them related as below:

“000”: idle

“001”: export result

“010”: no query result

“011”: buffer overflow

“111”: buffer busy

C. ARP control module functional simulation

1) simulate the query and send function of ARP request message

According to ARP standard, when receives MAC address query request, ARP will query the buffer. In common when there is no query result, ARP will send a broadcasting message just like figure 10. First we reset the system. Then at

250ns we import a query data, ARPctrl_in_source is “10” indicates that the query is coming from IP. As we can see from figure 11, the module immediately shift the ARPCTRL_BUF_CS and ARPCTRL_BUF_READ from bottom to high, and export IP address to the port of ARPctrl_buf_out. This indicates that ARP control module has export query request to buffer automatically. And about at 1800ns, buffer return a signal standing for no data found (because this IP is inexistent), we can see that ARPctrl_send_en turns into high level electricity immediately, at the same time the port of ARPctrl_send_frame sends out 42bytes request message to outside. Until now we validate the message according with ARP request message format and the front 6bytes of the message are broadcasting address, and the IP field includes the IP address which we have input before.

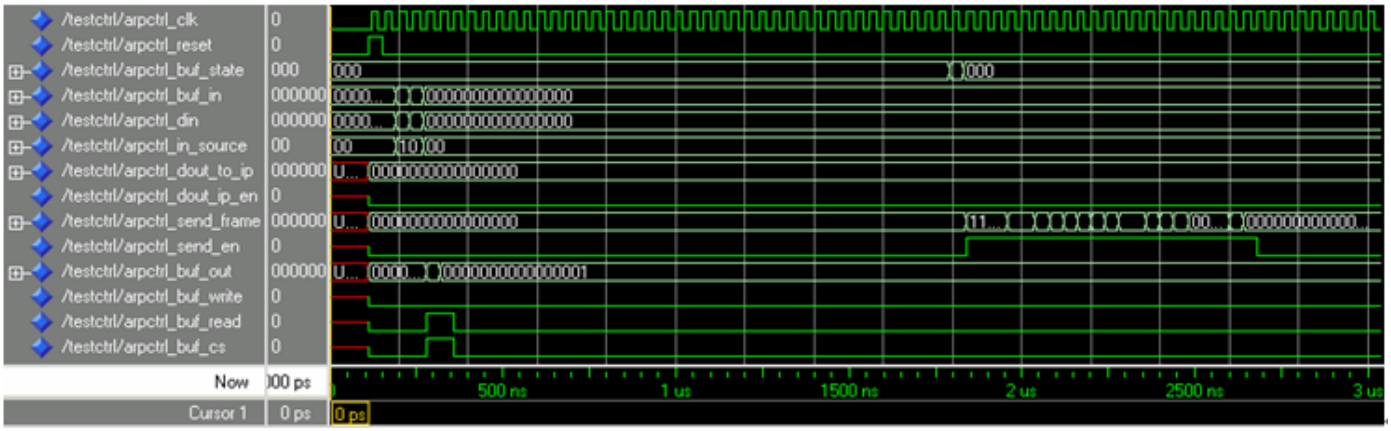

Figure 11. Waveform diagrams of query and sending message functional simulation (ARP control module)

Query result exports please see ARP buffer module simulation.

2) Simulation of receiving data frames about ARP control module

The test is showed in figure 12. Simulation process: first we input a virtual user-defined frame to module. In figure 8 we can see the signal of ARPctrl_in_source is “01” and this stands for outside ARP request or respond message. According to ARP standard, both ARP request and ARP respond will write the sending end information of MAC and IP to buffer. The test result indicate that ARP control module turns the ARPCTRL_BUF_CS and ARPCTRL_BUF_WRITE to high, and the port of ARPctrl_buf_out will export 5 cycles data to buffer. We validate that 10bytes data are the right which we had define the virtual frame of MAC and IP information in sending end, so these circumstances show us that the ARP control module can export address information to buffer automatically.

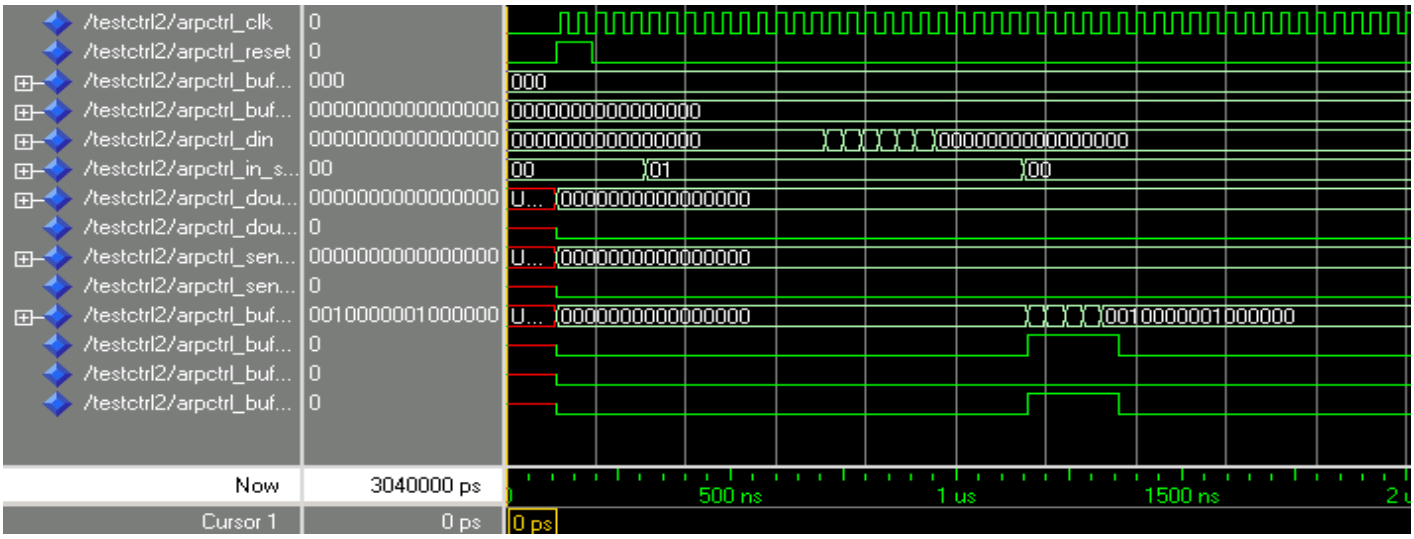

Figure 12. Waveform diagrams of receiving data frames simulation (ARP control module)

VI THE WHOLE ARP MODULE FUNCTION SIMULATION

Now we combine the two module and draw a top file link map just as figure 13, the left is ARP buffer module and the right is ARP control module. The ports of them and the related function have been introduced.

ARP module function simulation process：As figure 14, we import an ARP message to ARP module. According to ARP function, ARP module will put IP and MAC addresses into buffer (no matter how the frame is request or respond). After a while about 2us we query the right IP address thought ARP module, we can see that arp_data_out_ip exports the corresponding MAC address. At last about 3us, we import an inexistent IP address to query, and we can see at 3.5us, module exports an ARP request message. Until now we validate the whole ARP module function.

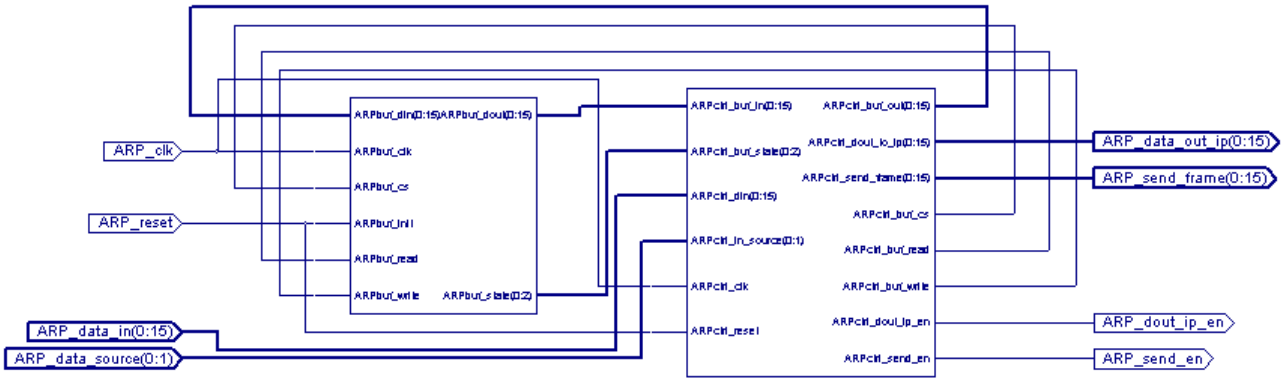

Figure 13. The file combined chart

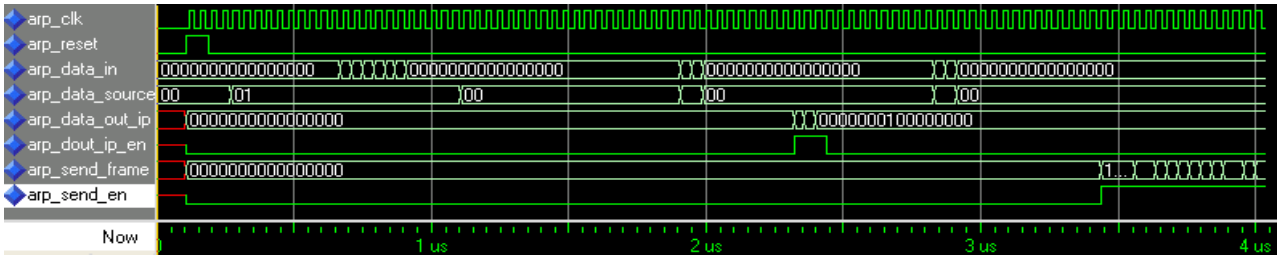

Figure 14. ARP module function simulation

VII. THE TEST OF ZERO-COPY AND CONCLUSION

The programming of test program can follow the policies below:

1) Detection whether the buffer contains valid data. The method is to test whether the curr_pack->len is 0 and its flag is READ.

2) Obtaining the data from buffer. The method is to write the buffer indicator to certain location, then the application retrieves the read location via mapping.

3) Looking up the buffer area to find the buffer with valid data, when the buffer is found, repeat the steps 1 and 2.

4) Using the mmap procedure to do memory mapping, and to allow application to access the kernel space directly.

5) After read the buffer, set the buffer to WRITE, and free the buffer space.

During the test, we can use the makefile file and the snull.h header file in the snull program which is contained in LDD3. The test procedure contains 3 steps:

1) Putting the files snull.c, snull.h, makefile, mymain.c into folder snull, and input the make command to compile them. If successul, we can get file snull.ko.

2) Compiling the file mymain.c by input command: gcc -o mymain mymain.c. If successful, we can find the executable file mymain. Now we get the loadable module snull.ko and test application mymain.

3) Executing command insmod ./snull.koto load module snull. If successful, we can use command lsmod to check the current using module of the operating system. Then we can execute the command: ./mymain to get test results. The test results indicate that the allocated buffer in kernel space can be accessed by user application with mapping. Zero-Copy removes the multiple memory copies and provides a useful resolution to resolve the bottleneck of computer system communication.

In the practice, we had used Intel Pentium IV-3.0G CPU 512M system memory and linux(Redhat)kernel-2.6.11. The SEND Processing time is from Socket to RJ45 and the RECEIVE processing time is from RJ45 to Socket as list in Figure 5. We tested the processing time of data transmission

through protocol stack, INET and TOE, in linux kernel. The test outcome: the average processing time for SEND through INET required 86.5us, average processing time through TOE with ML403 was 68.5us. Therefore, the result says that TOE method can offload nearly 20.8 percents of the processing cost in SEND to TOE. About RECEIVE the average processing time through INET was about 78.8us and 70.1us was required in the TOE. The result shows that TOE module can offload about 11.0 percents of the TCP/IP processing cost in RECEIVE. As like the result, TOE with ML403 method can offload about 10~20 percents of TCP/IP processing time in kernel to TOE device in the case of data transmission. So we can see the result is not perfect, the main reason is detail and process of design should improve on. But this kind of TOE method has a great potential to make offload percents improving.

In this paper, based on the analysis of the traditional message communication mechanism, we improved the current communication mechanism. By implementation of Zero-Copy technology in Linux system, we resolve the synchronization problem between user application and network card driver during the data transition. Meantime, by mapping amount of user application space to kernel DMA space, by modifying the network card driver interface to use application buffer directly, we reduced the message communication path, avoided the cost of memory copy and also dropped the CPU workload. Consequently we saved a lot of CPU time for application to do any complex computing, and resolved the bottleneck of whole system. This is very useful and has a realistic meaning to high capacity network.

The next further work is to design an effective and low delay protocol and evaluate its performance [11][12]. The work will involve the development of a high performance protocol stack, taking full advantages of hardware speed, and to implement the technology on a series of high speed network communication platforms.

ACKNOWLEDGMENT

This project is supported by Foundation of Liaoning Educational Committee in China (No. 2009A665) and Liaoning Provincial Natural Science Foundation of China (Grant No. 20102202) and Laboratory Chief Fund of Shenyang Normal University (No. SY200906).

REFERENCES

[1] Preparing for 2004－2005 Networking Ttransitions , Gary Gumanow, Carl Wilson, 2003

[2] I. Pratt and K. Fraser. Arsenic: a user-accessible gigabit ethernet interface. In Proceedings of Infocom, April 2001.

[3] P. Pietikainen. Hardware acceleration of Scheduled Transfer Protocol. http://oss.sgi.com/projects/stp.

[4] R. Martin, A. Vahdat, D. Culler, and T. Anderson. E_ects of communication latency, overhead, and bandwidth in a cluster architecture. In Proceedings of ISCA, June 1997.

[5] Jin-Soo Kim, Building a High Performance Communication Layer Over ,Virtual Interface Architecture on Linux Clusters,Acm,2001

[6] Efficient Operating System Support for Group Unicast, Martin Karsten, Jialin Song, ACM, 2005

[7] Linux Device Drivers 3 rd Edition by Jonathan Corbet, Greg Kroah-Hartman, Alessandro Rubini

[8] The Linux Kernel Module Programming Guide by Peter Jay Salzman, Michael Burian, Ori Pomerantz

[9] Ling Li, Liu Haipeng, Research of the Network Software Latency on Linux OS.

[10] Hsiao-Keng and Jerry Chu, Zero-Copy TCP in Solaris. Proc. of the USENIX 1996 Annual Technical Conference, Jan. 1996http://www.usenix.org.

[11] Mei-Ling Chiang and Yun-Chen Li, "LyraNET: A zero-copy TCP/IP protocol stack for embedded systems", Journal of Real-Time Systems, Springer Netherlands, Volume 34, Number 1, September, 2006, pp. 5-18

[12] S. Yamagiwa, K. Aoki and K. Wada, "Active Zero-copy: A performance study of non-deterministic messaging", Proceedings of the 4th International Symposium on Parallel and Distributed Computing (ISPDC’05), July, 2005, IEEE CS Press, pp. 325-332.

[13] W.Richard Stevens, TCP/IP Illustrated, Volume 1:The Protocol.

[14] David C. Plummer, An Ethernet Address Resolution Protocol-- or --Converting Network Protocol Addresses to 48.bit Ethernet Address for Transmission on Ethernet Hardware, RFC 826.

[15] Hou Boheng, Gu Xin, VHDL hardware describe language and numeral logic circuit design, University of Xin An Electron publishing company.

[16] Steve Golson，State machine design techniques for Verilog and VHDL.

[17] 2004 Vol.15 No.11 P.1689-1699Douglas L. Perry, VHDL--Programming by Example (4th Ed.) McGraw-Hill.