RESEARCH ARTICLE OPEN ACCESS

Design of NETWORK Device Driver with NAPI Implementation

G. Sivaram[1], B. Krishna[2], K. Prabhu[3]
[1](Assistant Professor in Department of Electronics and Communication Engineering, St.Martines Engineering college, Dhullapally, JNTUH)
[2] (Assistant Professor in Department of Electronics and Communication Engineering, KITE Women's College of Professional Engineering Science, Shabad, JNTUH)
[3](Assistant Professor in Department of Electronics and Communication Engineering, Maheswara Inst of Technology, Patancheru, JNTUH)

ABSTRACT
With the advent of extremely high speed Ethernet interfaces the number of interrupts the NIC receives is enormously high. To overcome this kind of problem NAPI mixes interrupts with polling and gives higher performance under high traffic load than the old approach, by reducing significantly the load on the CPU. NAPI is a technique to improve network performance on Linux. In this paper, the design of network device driver includes with NAPI implementation in Linux operating system based on ARM920T processor is implemented on the S3C2410-S development platform made in Beijing universal pioneering technology.
Keywords - ARM9 processor; embedded linux; network device driver; e100

I. INTRODUCTION

One of the many advantages of free operating systems, as typified by Linux, is that their internals are open for all to view. The Linux kernel remains a large and complex body of code. Often, device drivers provide that gateway. Network interface must register itself within specific kernel data structures in order to be invoked when packets are exchanged with the outside world by receiving packets asynchronously and has no such entry point in the /dev directory like block drivers. Network drivers also have to be prepared to support setting addresses, modifying transmission parameters, and maintaining traffic. The processor is interrupted for every packet received by your interface. In many cases, that is the desired Mode of operation, and it is not a problem. High-bandwidth interfaces, however, can receive thousands of packets per second. With that sort of interrupt load, the overall performance of the system can suffer. As a way of improving the performance of Linux on high-end systems, the networking subsystem need an alternative interface (called NAPI) based on polling.

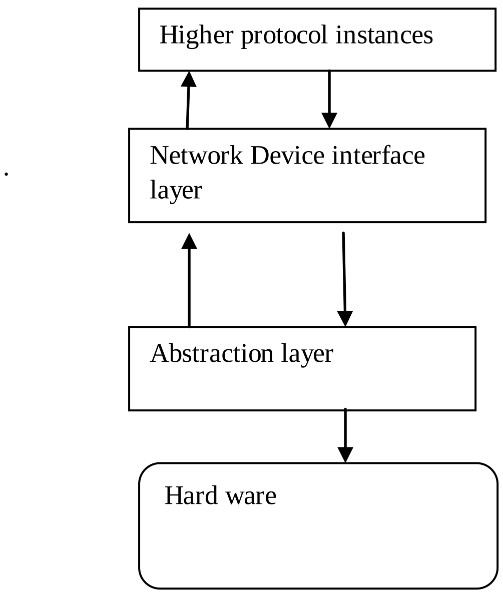
II. INTRODUCTION OF LINUX DEVICE DRIVER

In computing, a device driver is a computer program allowing higher-level computer programs to interact with a hardware device. A driver communicates with the device through the computer bus or Communications subsystem to which the hardware connects to the device. Once the device sends data back to the driver
The driver may invoke routines in the original calling program. Drivers are hardware-dependent and operating system specific. They usually provide the interrupt Handling required for any necessary asynchronous time-Dependent. Hardware interface. Typically Device drivers categorized into three types Depends upon the entry point in the /dev directory and also the Type of Data transfer.

1. Character device driver:
2. Block device driver
3. Network device driver

Character device driver having an entry point in the /dev directory. And also it can transfer one character at a time synchronously. Block device driver having an entry point in the /dev directory. And also it can transfer one character at a time .synchronously. Network Device Driver has no such entry point. The normal file operations (read, write, and so on) do not make sense when applied to network interfaces, so it is not possible to apply the UNIX "everything is a file" approach to them.

III. ARCHITECTURE OF NETWORK DEVICE DRIVER



Thus, network interfaces exist in their own namespace and export a different set of operations. Although you may object that applications use the *read* and *write* system calls when using sockets, those calls act on software object that is distinct from the interface. Several hundred sockets can be multiplexed on the same physical interface. Any network transaction is made through an interface, that is, a device that is able to exchange data with other hosts. Usually, an *interface* is a hardware device, but it might also be a pure software device, like the loopback interface. A network interface is in charge of sending and receiving data packets, driven by the network subsystem of the kernel, without knowing how individual transactions map to the actual packets being transmitted. Many network connections (especially those using TCP) are stream-oriented, but network devices are, usually, designed around the transmission and receipt of packets. A network driver knows nothing about individual connections; it only handles packets. Not being a stream-oriented device, a network interface isn't easily mapped to a node in the file system, as */dev/tty1* is. The UNIX way to provide access to interfaces is still by assigning a unique name to them (such as eth0),.Communication between the kernel and a network device driver is completely different from that used with char and block drivers. Instead of *read* and *write*, the kernel calls functions related to packet transmission. network device receive data is achieved through interrupts. Upon receiving the information, an interrupt is generated, the driver will apply a sk_buff (skb) in interrupt handler, and the data read from hardware is placed to applied buffer.A straightforward method of implementing a network driver is to interrupt the kernel by issuing an interrupt request (IRQ) for each and every incoming packet. However, servicing IRQs is costly in terms of processor resources and time. Therefore the straightforward implementation can be very inefficient in high-speed networks, constantly interrupting the kernel with the thousands of packets per second. Overall performance of the system as well as network throughput can suffer as a result Polling is an alternative to interrupt-based processing. The kernel can periodically check for the arrival of incoming network packets without being interrupted, which eliminates the overhead of interrupt processing. Establishing an optimal polling frequency is important, however. Too frequent polling wastes CPU resources by repeatedly checking for incoming packets that have not yet arrived. On the other hand, polling too infrequently introduces latency by reducing system reactivity to incoming packets, and it may result in the loss of packets if the incoming packet buffer fills up before being processed.

IV. ETHERNET CONTROLLER CHIP e1000

Provide too many feature to reduce kernel's work like Link auto-negotiation, Filters IP addresses, Compute IP packets' checksum, Tries to limit the number of interrupts, Provide effective shutdown method (a.k.a. *Packet of Death).*Full customization of LEDs' blinking!

V. NAPI ("New API"):

NAPI is a modification to the device driver packet processing framework, which is designed to improve the performance of high-speed networking. NAPI works through:

VI. INTERRUPT MITIGATION

High-speed networking can create thousands of interrupts per second, all of which tell the system something it already knew: it has lots of packets to process. NAPI allows drivers to run with (some) interrupts disabled during times of high traffic, with a corresponding decrease in system load.

VII. PACKET THROTTLING

When the system is overwhelmed and must drop packets, it's better if those packets are disposed of before much effort goes into processing them. NAPI-compliant drivers can often cause packets to be dropped in the network adaptor itself, before the kernel sees them at all.

VIII. ADVANTAGES OF USING NAPI

The load induced by interrupts is reduced even though the kernel has to poll. Packets are less likely to be re-ordered, while out of order packet handling might be a bottleneck otherwise. In case the kernel is unable to handle all incoming packets, the kernel does not have to do any work in order to drop them: they are simply overwritten in the network card's incoming ring buffer. Without NAPI, the kernel has to handle every incoming packet regardless of whether there is time to service it, which leads to thrashing.

IX. A DRIVER USING THE NAPI INTERFACE WILL WORK AS FOLLOWS:

Packet receive interrupts are disabled; the driver provides a poll method to the kernel. That method will fetch all incoming packets available, on the network card or a DMA ring, so that they will then be handled by the kernel. When allowed to, the kernel calls the device poll method to possibly handle many packets at once.

X. PERFORMANCE UNDER HIGH PACKET LOAD

NAPI provides an "inherent mitigation" which is bound by system capacity as can be seen from the following data collected by Robert Olsson's tests on Gigabit ethernet (e1000).

Psize	Ipps	Tput	Rxint	Txint	Done	Ndone
60	890000	409362	17	27622	7	6823
128	758150	464364	21	9301	10	7738
256	445632	774646	42	15507	21	12906
512	232666	994445	241292	19147	241192	1062
1024	119061	1000003	872519	19258	872511	0
1440	85193	1000003	946576	19505	946569	0

Observe that when the NIC receives 890Kpackets/sec only 17 rx interrupts are generated. The system can't handle the processing at 1 interrupt/packet at that load level. At lower rates on the other hand, rx interrupts go up and therefore the interrupt/packet ratio goes up (as observable from that table). So there is possibility that under low enough input, you get one poll call for each input packet caused by a single interrupt each time. And if the system can't handle interrupt per packet ratio of 1, then it will just have to chug along

XI. CONCLUSION

Normally Now a Days Speed is the most important criteria. In order to meet that we need to use Gigabyte Ethernet controller .Ethernet controller chip having more and more throughput .in order to provide that much throughput Network driver module needs NAPIApi

ACKNOWLEDGMENT

This research paper is made possible through the help and support from everyone, including: parents, teachers, family, friends, and in essence, all sentient beings.,
Especially, please allow me to dedicate my acknowledgment of gratitude toward the following significant advisors and contributors:
I sincerely thank to my parents, family, and friends, who provide the advice and financial support. The product of this research paper would not be possible without all of them.

References

[1] Q.Sun, Developing Detail Explain of Embedded Linux Application, Beijing, Posts & Telecom Press, July 2006.
[2] T.Z.Sun and W.J.Yuan, embedded design and Linux driver development guide, Beijing, Publishing House of Electronics Industry, October 2009.
[3] Jonathan Corbet, Alessandro Rubini, Greg Kroah-Hartman, Linux Device Drivers, 3rd Edition, O'Reilly Media, February 2005.
[4] D.Cao and K.Wang, "Research of Network Device Driver based on Linux", Computer Knowledge and Technology, 2005(21).
[5] F.J.Li and W.D.Jin,"Research and Implementation of Network Driver in Embedded Linux", Modern Electronic Technique, 2005(16)
[6] M. Barr, Anthony Massa, Programming Embedded Systems, second edition, O'Reilly, 2006.
[7] G. C. Buttazzo, Hard Real-Time Computing Systems:Predictable Scheduling Algorithms and Applications, Second Edition, Springer, 2005.
[8] David E. Simon, An Embedded Software Primer: Addison-Wesley, 2006.
[9] Karim Yaghmour, Building Embedded Linux Systems: Reilly, 2003.
[10] W. Wolf, Computers as Components: Principles of Embedded Computer Systems Design: Morgan Kaufman, 2000.

Authors

Mr G.Sivaram has 6 years of teaching experience and presently working as an Assistant Professor in the Department of ECE in ST. Martines EngineeringCollege, Dhullapally, JNTUH, Hyderabad, AP (India).He received his B.Tech degree in ECE from JNTU in 2007, M.Tech in Embedded systems from JNTU, Hyderabad. He has guided several M.Tech and B.Tech projects. His areas of interests are Communication Systems, VLSI System Design and Embedded Systems.

Mr B.Krishna has 7 years of teaching experience and presently working as an Associate Professor in the Department of ECE in KITE Women's College of Professional Engineering Science, Shabad, JNTUH, Hyderabad, AP (India).He received his B.Tech degree in ECE from JNTU in 2005, M.Tech in VLSI from JNTU, Hyderabad. He has guided several M.Tech and B.Tech projects. His areas of interests are Communication Systems, VLSI System Design and Embedded Systems.

Mr K.Prabhu has 7 years of teaching experience and presently working as an Associate Professor in the Department of ECE in MaheshwaraInstituteofTechnology, Patancheru, Hyderabad, AP (India).He received his B.Tech degree in ECE from JNTU in 2003, M.Tech in Systems and Signal Processing from JNTU, Hyderabad. He has guided several M.Tech and B.Tech projects. His areas of interests are Communication Systems, VLSI System Design and Embedded Systems.