Development of Device Drivers Embedded in Real Time OS for SPring-8 SR Control System

T.Masuda, S.Fujiwara, T.Nakamura, H.Takebe, and T.Wada
JAERI-RIKEN SPring-8 Project Team
SPring-8, Kamigori, Hyogo, 678-12, JAPAN
masuda@rkna50.riken.go.jp

Abstract

A distributed computer system has been adopted for the SPring-8 SR control system. For lower level computers, we intend to adopt VMEbus computer systems with the real time OS which are compliant with POSIX.

For R&D study, we introduced LynxOS and wrote device drivers for Digital Output(DO), Digital Input(DI) and Analog Input(AI) boards on VMEbus. They were successfully operated with device drivers.

I. INTRODUCTION

The control system of the SPring-8 storage ring (SR) is designed to have three levels of hierarchy such as presentation layer, processing layer and equipment layer. An open system based on international standards is preferable because of its expandability and maintainability [1,2]. In the presentation layer, Engineering Work Stations(EWSs) running UNIX OS and X-terminal are used. To select the UNIX provides a merit of vender independence. In the processing layer, VMEbus computer systems are used.

The selecting criteria of an OS for VMEbus computer system are having real time features and being compliant with POSIX(Portable Operating System Interface for computer environment). POSIX is a standard OS proposed by IEEE committee. Among such OS's, LynxOS and HP-RT are commercially available candidates [3].

Generally speaking, there are two ways to access and to control I/O boards from application programs. One is the way with "shared memory", and the other is with a "device driver". The merits of using shared memory are faster access speed and easy realization. However, using device driver is more preferable, because it has advantages as bellow.

- Hardware interrupt can be treated.
- Faster timer interruption can be treated.
- Application programs can be independent of a particular hardware.

We introduced LynxOS and HP-RT and wrote LynxOS device drivers for some I/O boards. We will write HP-RT device drivers in the near future.

II. FEATURES OF THE REAL TIME OS

A. LynxOS

LynxOS is a product of Lynx Real Time Systems Inc. and is one of the real time OS called "real time UNIX". It provides us with a superior environment for software development of UNIX such as GUI (Graphical User Interface) based on X window system, standard network system based on Ethernet and TCP/IP, and so on. Furthermore, it provides us with real time functions such as preemption and priority-based task scheduling which UNIX dose not have.

LynxOS is developed with emphasis on standardization and open system. It is binary level compatible with UNIX SystemV R.3 and Source level compatible with UNIX 4.3 BSD. As for the shell, in version 2.1 it supports bash, csh and improved sh shells as well as dlsh shell. Dlsh is its original shell.

LynxOS also complies with POSIX 1003.1 ,1003.4 and 1003.4a. POSIX 1003.1 is a main part of POSIX. It defines interface with application programs. POSIX 1003.4 and 1003.4a[1] are extensions for real time feature and define interface with real time application programs. Therefore users' programs on LynxOS are portable to other OS's which are compliant with POSIX1003.1 and 1003.4.

LynxOS supports many platforms such as i80386/486, M68030/40, i860, M88K, SPARC and R3000. The kernel is about 190KB. It is ROMable and can be used as an embedded system.

B.HP-RT

HP-RT, a product of Hewlett Packard Co., is based on LynxOS. Its kernel is modified and tuned for PA-RISC. The main difference of it from LynxOS is the development environment. LynxOS provides us with a

[1]The POSIX 1003.4 and 4a are currently drafts.

self-development environment, while HP-RT provides a cross-development environment. Loading software from host system to target system requires much labor of developers. When application programs are developed, HP-RT must be accompanied by a particular host system, HP9000 series 700 or 800. Although larger initial cost is required, HP-UX provides us with superior development supporting tools such as SoftBench.

III. DEVICE DRIVER DEVELOPMENT

Device drivers are embedded in the OS kernel and glue the kernel to devices such as VME I/O boards. Device drivers convert the commands from the kernel into those the devices can understand, and vice versa. A device driver is a collection of function routines called "entry points" only through which the kernel can access the driver.

A. Feature of the LynxOS device driver

A device driver contains eight entry points. The names of entry points and their purposes are shown in Table.1. Writing a device driver is to describe how each entry point should react when the entry point is accessed from the kernel.

Table 1. LynxOS device driver entry points and its purpose.

Entry Point	Purpose
open	called when device is opened
close	called when device is closed
read	called to read data
write	called to write data
select	support *select* system call
ioctl	device control
install	called to install major device
uninstall	called to remove major device

Many special library functions with C interface are provided for writing device drivers. They are called the "driver service calls". These functions support dynamic memory allocation, 1 msec and 10 msec timer interrupt, hardware interrupt dispatches, software interrupt, thread, semaphore, signal, DMA chain,etc. Device drivers can be written in C language.

B. Development Environment

Device drivers are embedded in the kernel. We have to modify the configuration table file and reboot the system to remake the kernel when a new device driver are added. This process is called "static loading" of the device driver. Static loading requires much labor during the development. Therefore powerful and useful functions called "dynamic loading" are provided for developers. Device drivers can be linked or unlinked to the kernel with some commands from shell. All things we need to do are merely to declare the entry points in the form of specific structure in the device driver. Neither to modify the configuration file nor to reboot the system are required. There are no performance penalty. When the development is finished, the device drivers may be loaded statically.

Debugging tools for the development of device drivers are poor. Before version 2.1, only the way to debug device drivers is to print the information on console with *cprintf* or *kprintf* functions . In version 2.1, *skdb(Simple Kernel level Debugger)* debugger is provided. We will test it though it seems that it dose not have sufficient power.

C. Hardware Interruption

We have implemented hardware interrupts from I/O boards with the device drivers. All things we have to do in the device driver are to describe an "interrupt software handler" and to establish a relation between the header address of "interrupt software handler" and interrupt vector number which is sent to CPU from the I/O device which demands an interruption. The latter is implemented with a driver service call related to the hardware interrupt dispatches in the install entry point. This service call has an interrupt vector number and address of an "interrupt software handler" as arguments. It is very easy to implement a hardware interrupt with this service call.

D. Some tests with developed device driver

We have developed LynxOS device drivers for three types of VME boards. They are DO, DI (with no hardware interruption) and AI (with hardware interruption). Generally speaking, since an AI board has more functions than DI/DO boards, making a device driver for an AI board is more complex. We developed a simple device driver for the AI board which supported the least functions we need. All developed device drivers work well, and support all functions we need.

We examined the processing time for analog data acquisition with the developed device driver. The conditions of these tests are using the AI board with

12bit ADC, 30 μsec sampling rate, taking 64 samples per one read cycle and acquiring data from a fixed channel.

The processing time were measured with *gettimeofday* system call which provides the current system time with the resolution of 10 msec. As the processing time to be measured is actually less than 10 msec, we measured the average processing time repeated 1000 or 10000 cycles.

a) comparison between device access ways

We compared two different device access ways, by using shared memory and by using a device driver. Both data acquisition modes were set to the polling mode In the polling mode, a flag which is set when 64 samplings are finished is always polled. One data acquisition cycle by using shared memory included getting shared memory, setting the device operation, reading the data and releasing shared memory. One data acquisition cycle with a device driver included opening the device, setting the device operation, reading the data and closing the device.

It took about 2.5 msec by using shared memory per one data acquisition cycle and about 16.5 msec by using a device driver. In the case of using device driver, the overhead time for opening and closing device are very large. The processing time required for setting and reading is about 2.5 msec. It is as fast as the processing time by using shared memory.

b) comparison between data acquisition modes

We compared two data acquisition modes, the polling mode and the hardware interrupt mode. The polling mode is described in a). In the hardware interrupt mode, an interrupt occurs when 64 samplings are finished.

It took about 16.5 msec in the polling mode and about 17 msec in the hardware interrupt mode per one data acquisition cycle. It seems that there is a little overhead time to process the hardware interrupt.

We compared the influence on the other process between data acquisition modes by polling mode and hardware interrupt mode. A data acquisition process and another process, looping process, were run at the same time. The data acquisition processes are the same ones as above.

The result is shown in Table.2. The time of data acquisition process includes the time of opening the device, 10000 cycles of setting the board and reading the data, and closing the device. It shows that data acquisition in polling mode influences largely looping process.

Table 2. The processing time of the data acquisition process and the loop process when they run at the same time. The numbers in the parenthesis are processing times when they run alone.

data acquisition mode	data acquisition process	looping process
polling	47.8 sec (24.8 sec)	45.8 sec (23.0 sec)
H/W interrupt	53.0 sec (30.0 sec)	23.0 sec (23.0 sec)

IV. SUMMARY

We wrote and examined the LynxOS device drivers. We also intend to write those of HP-RT and test their compatibility with LynxOS.

V. REFERENCES

[1] T. Wada: "Design of the Control System for the SPring-8", *RIKEN Accel.,Prog. Rep.*, **24**, 202 (1990).

[2] T.Wada,T.Kumahara,H.Yonehara,H.Yoshikawa, T.Masuda, and Wang Zhen: "Design of SPring-8 Control System", *Proceedings of the International Conference on Accelerator and Large Experimental Physics Control Systems*, KEK, Tsukuba, Japan, November, 1991, pp. 151-153

[3] T.Masuda, T.Nakamura, T.Wada and Z.Wang: "The Real Time Operating System of Front-End Processors for the SPring-8", *RIKEN Accel.,Prog. Rep.*, **25**, 240 (1991).