PORTING EPICS TO L4-LINUX BASED SYSTEM

J. Odagiri, N. Yamamoto and T. Katoh KEK, Oho 1-1, Tsukuba, Japan

Abstract

Experimental Physics and Industrial Control System (EPICS) is now widely used for many accelerator control systems. While the current and the former versions of EPICS have required VxWorks to run core software on Input/Output Controllers (IOCs), the next version (R3.14) is to be portable to many other platforms.

Considering the recent trend toward Linux, it is an attractive candidate for the port. However, the Linux kernel cannot ensure real-time responsiveness because it dose not preempt the execution from a process that is running in the kernel. As an alternative, we adopted L4-Linux, a port of Linux onto a real-time micro-kernel (L4), as the platform. With some adaptation, L4-Linux allows any EPICS thread to benefit from either the real-time scheduling by L4 or the many functions of Linux.

The adaptation of L4-Linux to the real-time use, the interface libraries between the IOC software and L4-Linux, and a library to support the VMEbus are described. A preliminary result of the measurement of interrupt latency is also presented.

1 INTRODUCTION

EPICS consists of a set of software components and tools that application developers use to create a control system [1]. The basic components are:

- Operator Interface (OPI) – a UNIX based wrokstation which can run various EPICS tools
- IOC – a front-end computer containing various I/O modules and interface modules for extending other I/O buses
- Local Area Network – the communication network which allows the IOC and OPIs to communicate.

While the architecture makes EPICS scalable, it is expensive to adopt EPICS for very small control systems. If the workstation and the IOCs can be replaced with a single PC, EPICS based system becomes a cost-effective solution for a small control system.

Considering the recent progress of Linux, it seems to be a promising platform for the PC based EPICS. Since the OPI tools run under UNIX, there should not be essential difficulties in porting them to Linux. On the other hand, the IOC software (iocCore) was developed based on VxWorks real-time operating system [2]. In the next version of EPICS R3.14, however, VxWorks application interfaces are to be isolated, and then, to be replaced with Operating System (OS) interface libraries [3]. With the implementation of the OS-interface libraries provided, iocCore can run on multiple platforms. In fact, the first release of R3.14 comes with the OS-interface libraries for VxWorks, Linux and some other operating systems.

In the next section, the limitation of Linux as a real-time platform for iocCore is discussed. A solution based on L4-Linux, a derivative of Linux, is discussed in the later sections.

2 IOC SOFTWARE AND LINUX

EPICS iocCore has layered structure centered on the run-time database, which is essentially a snapshot of the I/O channels of the devices under IOC's control, as illustrated in Fig. 1.

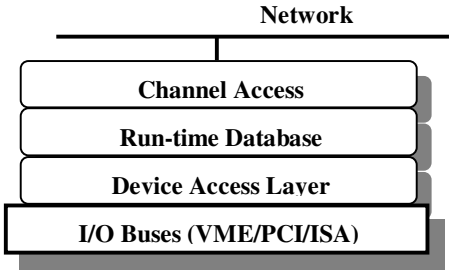

Figure 1: Structure of iocCore

Above the run-time database, Channel Access (CA) works as a "software bus" to communicate with the OPI tools and other IOCs on the network. Below the run-time database is the device access layer that faces the hardware modules on the I/O buses. In order to support the activities at both of the layers, a set of co-operative threads work together by sharing global variables in a single address space. The OS for iocCore thus must provide threads that run in a user address space.

Since iocCore is a real-time system, the threads must be scheduled by the urgency of their work. It applies particularly to the threads of the device access layer. Some data in a device may be lost if a thread that handles the I/O operation responds late, or a software sequencer may be running to control a device locally requiring deterministic execution. The OS must schedule the threads by their fixed priorities to meet the requirement. In addition, the OS must ensure that the urgent threads are not blocked by less urgent activities.

Unfortunately, the Linux kernel allows usual processes to block an urgent process for considerable durations of time. A process that performs IDE disk I/O can block an urgent process for several tens of milliseconds [4]. The possible durations of the blocking in general can be over 130 milliseconds on a 266 MHz class Pentium II processor [5]. The main cause of the blocking is the non-preemptive-ness of the Linux kernel. Once a process has entered the kernel by issuing a system call, any other process that gets ready to run is forced to wait for the running process to go through the kernel. It applies even if the process newly scheduled has a higher priority. In order to reduce the durations of the blocking, it is necessary to modify the Linux kernel so that the process running in the kernel invokes the scheduler frequently giving a possible urgent process more chances to run. An alternative of the approach is to bring an underlying framework into the system in order to enable the preemption to make [6], as discussed in the next section.

3 IOC SOFTWARE ON L4-LINUX

3.1 What is L4-Linux?

L4-Linux was developed at Dresden Institute of Technology in corporation with IBM Watson Research Center [7]. It is a port of Linux kernel as a server task on top of a real-time micro kernel named L4, or its successor named Fiasco [8], as illustrated in Fig. 2.

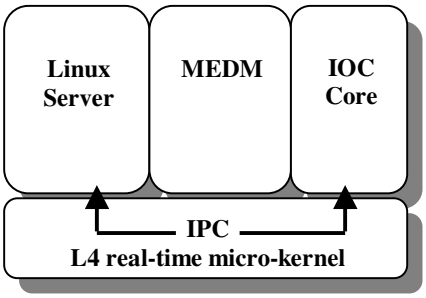

Figure 2: EPICS on L4-Linux

L4 is a preemptive micro kernel, which provides its tasks with the only three primitives, threads, address spaces, and Inter Process Communication (IPC). Each L4 task has its own address space in which up to 128 of threads can run.

Linux processes are implemented by putting the first thread of a L4 task to use for the "main". The Linux processes call the Linux server for a service through an IPC call. L4-Linux is compatible with standard Linux at the binary level, provided with an emulation library that transforms a Linux system trap (INT 0x80) into the IPC call. A page fault of a process is also transformed into an IPC with the Linux server. The Linux server manages the page fault the same way as standard Linux does.

Every Linux process, being a L4 task, can also issue L4 system calls.

3.2 L4-Linux as a real-time platform

L4-Linux was designed as a partner of a real-time system that is running on the same computer, and hence it in itself is not a real-time system. However, if a process of L4-Linux is given a priority higher than that of Linux server, it can preempt the execution from the Linux server, because both the process and the Linux server are independent tasks under L4's scheduling. Assuming the preemption has happened, if the process calls Linux for a service, it has to be blocked waiting for Linux to get the previous work done and accept the call. On the other hand, if the process does not rely on Linux, it can continue with its work under L4's scheduling.

The latter case applies to a thread that performs I/O at the device access layer of iocCore. It just does something primitive when it is triggered by an interrupt, such as reading or writing data from/to the registers of the device, moving the data from some memory address to another, operating semaphores to notify the event to another thread, and so on. They can get the urgent work done without calling Linux, hence without being blocked by the Linux server. On the contrary, the CA relies on Linux for the TCP/IP socket services. If a thread that works for CA has preempted the Linux server, the thread ends up returning the execution to Linux when it calls Linux for the service. Consequently, the preemption makes a difference only for the threads of the device access layer, where high responsiveness is really needed.

Once the CPU has accepted an interrupt that triggers an urgent thread, the L4 kernel switches the execution to the urgent thread in a predictable time. However, the interrupt itself can be blocked because the Linux server has critical sections, which are protected by disabling interrupts. If the interrupt comes in when the Linux server is executing one of the critical sections, the preemption is delayed until the execution leaves the critical section. The standard Linux kernel disables interrupts for durations up to several hundreds of microseconds [6]. The duration can be even longer in special cases [5]. It also applies to the Linux server of L4-Linux. In order to ensure the pre-emption to happen within 100 microseconds or less, it is necessary to adopt another method for the protection of the critical sections. In L4-Linux, a "Linux interrupt handler" is just another thread of the Linux server task. A real interrupt handler in the L4 kernel schedules the Linux interrupt handler when the interrupt occurs. All of the Linux interrupt handlers as well as the Linux server can be blocked without disabling interrupts as a result. The critical sections can be protected by just raising the priority of the thread that is executing a critical section. The replacement of the method to protect the critical sections is an improvement to be done in the future.

3.3 Required Modifications

The main modification required to launch real-time threads was to allow them to have a priority higher than that of the Linux server. In addition to it, another modification was required in relation to the virtual memory management.

In L4-Linux, there are two different sets of page tables for a virtual memory space of a process. One is in the Linux server and the other is in the L4 kernel. The former is the one Linux manipulates to decide how it uses memory. It is logical in the sense that it is not connected to the Memory Management Unit (MMU). The latter is a set of physical page tables that the MMU refers to. It is empty when a process is created. Every time the process causes a page fault, an entry of a physical page table is updated by referring the corresponding logical page table, making the both sets of tables equivalent.

The behaviour of the memory management becomes somehow different with standard Linux, for example, in case a process has issued the "mlockall" system call to make itself memory-resident. After the pages have been swapped in, and the logical page tables have been modified, the system call returns leaving the physical page tables unchanged. The process still can cause page

faults just to update the physical page tables. This does not matter as long as the system is used as a time-sharing system. However, the IPCs associated with the page faults break the assumption that the real-time threads do not rely on the Linux server while they are working on their urgent work. In order to cope with the problem, an L4-Linux specific system call is created to make Linux flush the entries of the logical page tables down to the physical ones.

3.4 OS-interface libraries for L4-Linux

This subsection describes our implementation of some of the EPICS OS-interface libraries for L4-Linux.

The thread library provides iocCore with a multi-thread environment that schedules threads by their fixed priority. As mentioned earlier, an L4 task can have up to 128 threads in its address space. It seems to be efficient to put these threads to use for the library. Unfortunately, this scheme does not work because the Linux server does not distinguish a thread from the others in an L4 task. It implies that only one thread in an L4 task can call the Linux server at a time. Instead, the "clone" system call is available to create "Linux threads". In this case, a Linux thread corresponds to an L4 task, which has its own address space, i.e. the physical page tables. Created through the clone system call, a Linux thread shares the page tables with its creator in the Linux server, not in the L4 kernel. The Linux thread becomes really a clone of its creator when it fills out its own physical page tables by referring the shared logical page tables.

The semaphore library provides binary semaphores and mutual exclusion semaphores. Since the IPC that L4 offers is synchronous type, semaphores can be implemented on top of the IPC. A thread sleeps by waiting an IPC message when it tries to take an empty semaphore. Another thread wakes it up by sending the IPC message when it gives the semaphore. The IPC can also take on the timeout handling of the semaphore operations. Critical sections in the implementation need to be protected by disabling interrupts.

In the libraries, functions were implemented by using only L4 system calls as far as real-time threads make regular use of them. The other functions supposed to be used only in the initialization step may invoke some of the Linux system calls.

4 VME-BUS SUPPORT

The main motivation of this port was to run EPICS on PCs for small control systems. On the other hand, if the system can run on VME board computers, it can be also used for large scaled control systems. In fact, VME single board computers compliant with the PC specifications have been released from several manufactures [9]. In addition, a Linux device driver for the Universe PCI/VME bridge chip was developed by Hannappel [10]. These circumstances encouraged us to port the L4-Linux based EPICS to a VME CPU board. For this purpose, a VME support library was developed based on the OS-interface libraries and the Universe driver. Together with the OS-interface libraries, the VME support library provides essential functions required to port the existing EPICS drivers to the system. A thread created in the ported drivers is to run as a Linux thread, which was described in the previous section.

5 MEASUREMENT OF LATENCY

To confirm the validity of the scheme, interrupt latency was measured on a VME CPU board, which has a Celeron 300 MHz processor. A process that causes heavy I/O load on an IDE disk was used as a background [4]. With this background running, a thread got the CPU clock-count and issued a command to a VME module, which caused the interrupt. Triggered by the interrupt, an interrupt handler (another thread connected to the interrupt) got the CPU clock-count again and cleared the interrupt. Iterating the above steps, the latency, the difference of the two clock-count values, was measured. The latency in the worst case was found to be about 800 microseconds in 10^5 times of trials.

6 CONCLUSIONS

As a platform for the PC-based EPICS, L4-Linux was adopted and modified to launch real-time threads. The essential functions of both OS-interface libraries and a library to support the VMEbus were implemented.

By using the libraries, interrupt latency was measured on a PC compliant VME CPU board. The latency in the worst case was less than one millisecond. It indicates that the real-time threads actually preempted the execution from the Linux server. The dominant factor of the latency should be the critical sections executed by the Linux server with disabling interrupts.

7 ACKNOWLEGEMENTS

One of the authors, J. Odagiri, would like to thank Y. Yasu and K. Nakayoshi of the online group at KEK for their helpful suggestions on the design of the VMEbus support library and the method to measure the interrupt latency.

8 REFERENCES

[1] http://www.aps.anl.gov/epics/
[2] http://www.windriver.com/
[3] M. Kraimer et.al., "EPICS: Porting iocCore to Multiple Operating Systems," ICALEPCS'99, Trieste, Italy, Oct. 1999.
[4] http://www-online.kek.jp/~nakayosi/
[5] http://www.mvista.com/realtime/
[6] http://www.rtlinux.org/
[7] http://os.inf.tu-dresden.de/L4/LinuxOnL4/
[8] http://os.inf.tu-dresden.de/fiasco/
[9] http://www.vmic.com/
[10] http://lisa2.physik.uni-bonn.de/~hannappe/