LINUX AND RT-LINUX FOR ACCELERATOR CONTROL - PROS AND CONS, APPLICATION AND POSITIVE EXPERIENCE

F. Nedeoglo, D. Komissarov, Department of Physics, Moscow State University, Russia
A. Chepurnov, Institute of Nuclear Physics, Moscow State University, 119899, Moscow, Russia

Abstract

Since 1995 we have actively been searching a possibility to apply the Linux operating system in particle accelerator control systems. Linux is a Unix-like operating system, freely distributed on GNU Public License (GPL), a development which has been underway since 1991 by several independent developer-enthusiasts. At present Linux is a stable operating system with plenty of possibilities, its popularity is growing fast. To support man-machine interface, we have used this operating system at top level in control systems for several projects of small electron linacs. In view of development convenience it would be desirable to use this operating system at a front-end real-time level, too. Therefore we looked for a possibility of creating applications under RT-Linux (real-time extension to Linux).

1 INTRODUCTION

PCs running under the Linux operating system (OS) have become very popular among developers of control systems (cs) for particle accelerators. The object-oriented cs TACO, for example, supports Linux OS [1]. The reason is that a PC together with Linux OS provides an adequate level of computational resources and reliability whereas the price-to-performance ratio is acceptable.

Most of the widely spread commercial software for PCs is not reliable enough and resource consumption from the stand-point of control applications is not acceptable, except for a few operating systems developed especially for control and real-time.

The Linux OS presents an attractive alternative to other OSs traditionally used in cs. Linux is dynamically developing as a modern Unix-like OS. It is freely available together with its source codes and well-documented. The main features of Linux are POSIX compatibility, GPL agreements and good network connectivity.

2 LINUX AT TOP LEVEL OF A CS

2.1 Upgrading of RTM injector cs

We used Linux OS to upgrade a rather old cs of the race track microtron (RTM) injector for the first time in 1996-97 [2,3]. The cs was built as a star-shaped network of LSI-11 compatible minicomputers. It consisted of four levels with six single-board minicomputers (Figure 1).

The equipment of the front-end level was connected to the control stations via a few CAMAC crates. The network concentrator machine and the control stations have no any external devices to store control programs. All computers of the cs communicated with each other through the network concentrator via optically isolated RS-232C links.

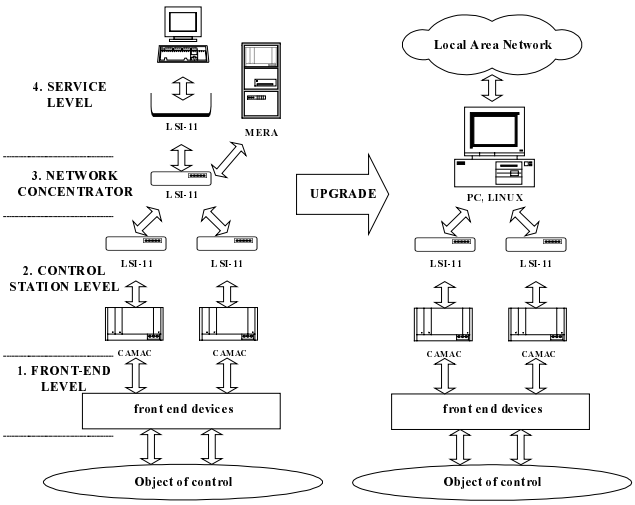

Figure 1. Upgrading of RTM injector control system

The operational software of the control stations was developed as a real-time system with digital feedback loops, control algorithms and re-entrant data acquisition drivers. Some computers of the service level maintained the man-machine interface and the database. To load the software into the control stations and the network concentrator the minicomputer (MERA) equipped with a hard disk drive was used.

During the upgrade process (Figure 1), all the computers of the service level and the network concentrator were replaced by a single PC running under Linux OS. The software developed for Linux OS emulates the functionality of the old top level completely. It allows us to load and use local software of the control stations as is. Several computers of the old generation were removed from the cs, which improved the reliability, reduced the maintenance time and increased the comfort of the operator. In addition, the Linux software enables us to

control or watch the control processes remotely via TCP/IP.

2.2 Cs for compact electron linac

A new cs was developed for the new small cw linac. It was the first accelerator, which belonged to a new family of industrial cw linacs [4]. The development of the cs had two stages. During the first stage of the cs development, flexible modern DAQ boards were installed in a conventional PC to control subsystems of the accelerator. In the second and final stage the cs will be based on CAN-bus to communicate with intelligent controllers belonging to the family of "Smart Devices" [5] which form the front-end level. The Linux kernel driver for the CAN-bus ISA adapter has been developed. We selected the DeviceNet high level protocol for the CAN-bus and developed a DeviceNet protocol stack for Linux OS [6].

The application software of the top level of the cs is based on the distributed shared memory (DSM) architecture. Applying the DSM approach provides independence of program modules from each other. It allowed us to use the same top level software during the both stages. The top level is implemented on the PC under Linux OS. The application software of the top level was designed with the help of GNU development tools only. The GNU C compiler (gcc) with a standard GNU C library (gnu libc) and the LessTiff widget set (Motif 1.2 compatible) were used.

3 REAL-TIME TASKS UNDER LINUX

The Linux scheduler is optimized for the balancing of response time and throughput [7]. The execution time of the process depends on the system load and the behavior of other processes in a complex fashion. Furthermore, loss of hardware interrupts may occure when Linux disables interrupts during the execution of critical kernel code segments. These reasons make Linux unsuitable to implement control algorithms, which require a predictable system response time. But there are decisions and projects underway aimed to improve Linux to be applicable for real-time (rt) tasks. The most known and widespread of them is Real-Time Linux or RT-Linux [8]. RT-Linux is based on an approach where a small rt kernel coexists with the usual Linux kernel. The rt processes are implemented as light-weight threads which are executed in their own address space. The Linux kernel operates as rt process with the lowest priority using a virtual machine layer in RT-Linux. Initially all hardware interrupts are handled by the rt kernel and are passed to Linux tasks only, if there are no real-time tasks to run. When Linux disables interrupts the emulation software (that was added to the Linux kernel at RT-Linux installation) will queue interrupts that have been passed on by the real-time kernel.

The rt kernel provides only those basic rt services which Linux can not provide. There is no support for data displaying, network access and other complex tasks in RT-Linux. Thus, a rt application consists of two parts. The first one is the rt task which incorporated in the loadable kernel module that implements the rt algorithm. The second part represents the Linux user process that takes care of functions that are not time critical. The rt tasks and Linux user tasks communicate through lock-free queues and shared memory. Lock-free queues are organized as FIFO (First In First Out) queue and are accessible by a Linux process via POSIX system calls such as *read/write/open/ioctl*. Shared memory is accessible via the POSIX *mmap* call.

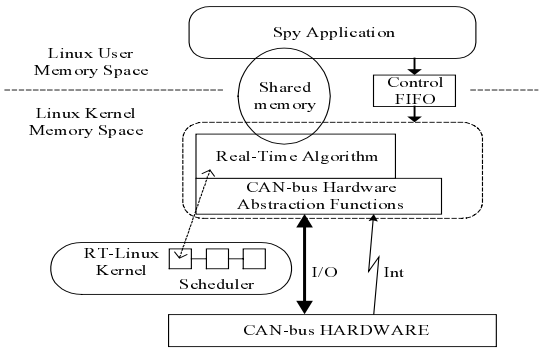

Figure 2. Real-time CAN-bus software module.

A skeleton of the real-time software module for the CAN-bus adapter was designed to try to use RT-Linux at the front-end level. The central idea of this software module is to implement the control algorithm in such a way that it runs independently from the top level software, but it is possible to control the flow of it and observe the data it processes.

The rt software module for the CAN-bus consists of a set of functions to work with the CAN-bus adapter, of the implementation of the particular rt algorithm and of a "spy" application. The functions to work with the CAN-bus adapter look like POSIX I/O calls (i.e. *read/write/open/ioctl*) and are used by the rt algorithm to access the CAN-bus network. They form a software layer, which hides low-level operations with the CAN-bus adapter from the rt algorithm. The implementation of the real-time algorithm together with the CAN-bus hardware abstraction functions are integrated into the loadable kernel module which works in kernel memory space. The "spy" application is an ordinary Linux process that transfers data from the shared memory to the top level applications or displays this data on the screen. It works in user memory space. The flow of execution of the rt algorithm is controlled by the “spy” application via the FIFO. It should be noted that the "spy" application does not change data in shared memory that are processed by the rt algorithm. So the shared memory acts as a "window" through which the "spy" application observes data being processed.

The DeviceNet application library allows to design a rt CAN-bus module which can access the CAN-bus as a

node of the DeviceNet network. But additional research of compatibility between the RT-Linux and DeviceNet application library should be done to apply the DeviceNet rt module in a real control task.

4 PROS AND CONS

Our four years experience has shown, that Linux OS provides a level of sufficient reliability and productivity to implement a man-machine interface, a client-server, a network and control applications. A capability of programming software for top level and real-time tasks (with the help of RT-Linux) allows to use Linux OS at all the levels of the cs where OS is needed. It is possible to point out good connectivity with different types of networks and network file-systems among other positive features of Linux. Linux provides a necessary set of development tools and program components to develop cs software. There is a wide spectrum of compilers, cross-platform libraries and tools to develop application software for this OS.

There is a number of DBMSs available for Linux OS which could be chosen to implement a database for the cs. PostgreSQL is well known and is a full-function DBMS available freely for Linux. Several commercial vendors of DBMSs have declared support of Linux OS by their products and some of these DBMSs are used in the cs under Linux already [9].

CORBA (Common Object Request Broker Architecture) is one of the object-oriented environments for distributed systems which provides the ability to build distributed applications, running on heterogeneous systems. CORBA is considered a promising technology for cs development [10,11]. There are some CORBA implementations for Linux both commercial and freely distributed. We have got some experience with MICO [12], which is an implementation of the CORBA 2.1 specification with GPL license agreements. However, it is necessary to make additional researches to find out MICO feasibility.

However, Linux OS still has gaps in some fields of the application software in spite of growing support by the software manufacturers. For example, Linux has no SCADA-type software. We have not found any ready-to-use implementation of the DeviceNet protocol stack for Linux. That is why we have developed our own DeviceNet application library.

Linux OS is true open software with regard to distribution as well as to development. The disadvantage of this fact is that only a few hardware manufacturers support Linux as target OS for their latest hardware. So, hardware drivers for Linux appear with some delay.

5 CONCLUSIONS

Linux can be a basic operating system to develop control systems for accelerators in spite of disadvantages the number of which decreases every day. It provides a number of software tools and has all the required features to design software such as man-machine interfaces, client-server network applications, control applications and databases. To control front-end devices the RT-Linux extension of Linux could be applied. Whereas a combination of Linux and RT-Linux provides both a development and a run-time platform it could be applied in all the levels of a cs where an OS is needed in.

The rt module to implement rt tasks with access to the CAN-bus was designed. Testing of this module with a DeviceNet application library is planned.

An additional investigation and improvement of the real- time compatibility and CORBA productivity should be done for Linux OS.

REFERENCES

[1] A.Gotz, W-D.Klotz, et al. “TACO: An object oriented control system for PCs running Linux, Windows NT”, CD-ROM Proc. of PCaPAC, October, 1996, DESY, Humburg, Germany.

[2] A.S. Alimov, A.S. Chepurnov, et al., "Perfomance of the 6 MeV Injector for the Moscow Racetrack Microtron.", Nucl. Instr. Meth. A326 (1993) 391.

[3] A.S. Chepurnov, I.V. Gribov, et. al., " Moscow University Racetrack Microtron Control System: Ideas and Developments", Proc. of ICALEPCS (Tsukuba, Japan, KEK, 11-15 Nov., 1991) Tsukuba, KEK, 1993, pp. 140-142.

[4] A. Chepurnov, A. Alimov, et. al., "Control System for New Compact Electron Linac.", These proc.

[5] A.S.Chepurnov, A.A.Dorochin, K.A.Gudkov, V.E.Mnuskin, A.V.Shumakov, “Family of Smart Devices on the base of DSP for Accelerator Control.”, Proc. of ICALEPCS, W2B-d (Chicago, Illinois USA,1995).

[6] A. Chepurnov, D. Komissarov, F. Nedeoglo, A. Nikolaev, "DeviceNet Implementations under Linux for Use in Control System of a Particle Accelerator." These proc.

[7] J. Epplin, "Linux as an Embedded Operating System", http://www.embedded.com/97/fe39710.htm

[8] RT-Linux Manual Project, http://www.rtlinux.org/~rtlinux/manual/RTLManual/RTLManual.html

[9] A.Yamashita, T.Fukui, M.Kodera, T.Masuda, R.Tanaka, Mikazuki, “RDBMS on Linux for accelerator control”, Proc. of PCaPAC’99 http://conference.kek.jp/PCaPAC99/cdrom/paper/poster/p51.pdf

[10] S. Hunt, B. Jeram, M. Plesko, C. Watson, "The Implementation of an OO Control System API with CORBA.", Proc. of ICALEPCS'97, Science Press, 1998, p. 354.

[11] The MICO CORBA Compliant System, http://www.mico.org/