ROBOT CONTROL USING REAL-TIME LINUX

Pavel Andris
Institute of Control Theory and Robotics
Slovak Academy of Science
Dúbravská cesta 9, 842 37 Bratislava, Slovakia
E-mail: utrrandr@savba.sk

Abstract

Robot control software is a typical application applying real-time Linux as a hard real-time operating system. The author's software was originally running under MS-DOS. Problems of adapting it for real-time Linux are presented. This paper describes internal structure of the software as it had to be divided into a hard real-time part (a kernel module) and a standard Linux process part.

1 Introduction

Institute of Control Theory and Robotics has developed an Integrated Programming System for Robot Cells Control. The system is used for computer control of flexible production systems and robot cells [1].

It is adaptable to various kinds of the robot cells hardware (control panels, servo-drives, robot effectors with various kinematic structures). A simultaneous control of several effectors is possible. ROBOL, a higher robot programming language, is implemented in the system. It is suitable for programming of a large number of various technological applications of robot cells, e.g. welding in a protective atmosphere, assembly, manipulation, laser or water ray cutting, gluing etc.

As the official name of the system is rather long, the name ROBOL is used either for the robot activity programming language or for the whole Integrated Programming System. Only the latter meaning is referred to in this paper.

In 1997–1998 ROBOL was implemented on a standard industrial robot PUMA 560. To avoid major hardware work, some ideas and software from RCCL/RCI robot control package [7, 6, 8] were used. Network interface was added to allow sending a model of a controlled robot workcell to a viewer (a remote computer). Another remote computer can be used to generate commands and send them to ROBOL.

ROBOL was originally implemented under MS-DOS. This simple operating system has good real-time capabilities. A user task has unrestricted access to the hardware and there is no multitasking. Our team wrote a simple thread scheduler and clock interrupt service routine. Anyway, the MS-DOS 640 kilobytes barrier could not be accepted anymore by quickly growing ROBOL.

The first new operating system to implement ROBOL under, that was considered, was MS-Windows 95. It was soon abandoned because

- it provides only soft real-time
- it would be necessary to buy some additional software development packages for device drivers because ROBOL uses some custom developed hardware

We have opted for Linux because any commercial hard real-time system was beyond our budget. Standard Linux provides soft real-time only, as well. ROBOL is a low frequency (14.3 or 35.7 Hz) application with plenty of floating point operations during each period. One period computations may take up to 3 milliseconds.

Soft real-time experiments only confirmed the old truth: it mostly works but occasional missing of deadlines has to be expected, especially if the control computer is used also as a normal workstation. In the robot world, missed deadlines stop robot and a human intervention is required. It was time to look for a hard real-time solution.

2 Hard Real-Time Linux

Linux is a free Unix-type operating system available for many platforms. Development teams all over the world are developing some modifications to the Linux kernel in order to provide a hard real-time operating system, e.g. NMT/FSMLab's RTLinux [15, 12], DIAPM's RTAI [9, 11], and the real-time system KURT [10, 5], all being well appropriate for control applications or even embedded systems. (NMT — New Mexico Technology, FSMLabs — Finite State Machine Labs, Inc., RTLinux — (hard) Real-Time Linux, DIAPM RTAI — Dipartimento di Ingegneria Aerospaziale - Politecnico di Milano Real-Time Application Interface, KURT — Kansas University Real-Time (Linux)).

Comparing the individual packages is beyond the scope of this paper. ROBOL has been implemented under RTLinux (RTL) [12].

RTLinux is a small, deterministic, real-time operating system that is somewhat like a single POSIX process sitting on a bare machine. Hard real-time applications are threads and signal/interrupt handlers in this process. Linux runs as the lowest priority thread of the RTLinux kernel and it is made always preemptible. This makes things simple, deterministic and fast.

The price to pay is that almost every real-time application has to be divided into two parts:

- Real-time part with strict timing requirements. It runs under RTLinux small, real-time operating system. It may consist of one or more RT threads and/or interrupt service routines. RT threads are scheduled according to their priorities. The term „RT (real-time) thread" is used to distinguish it from Linux POSIX threads. RT threads run in kernel space and memory context. This means that sophisticated services provided by Linux and even by C libraries are generally not available for the RT threads. RTLinux and RT threads are loadable kernel modules.

- Anything without strict timing requirements runs as a normal Linux process.

Both the parts need special tools to communicate because they run in different memory contexts. RT threads run in Linux kernel memory context and Linux processes in their own one. RTfifos provide a device interface that can be read/written on the Linux process side and written/read on the RT thread side. Shared memory utility is another tool.

3 ROBOL Controls PUMA

Some ideas and software from RCCL/RCI robot control software package were used to control a standard industrial robot (PUMA UNIMATION 560) by ROBOL [7, 6, 8]. The system architecture is shown on Fig. 1.

A bidirectional parallel connection between the ROBOL PC and the robot controller has been established. A RCCL/RCI program called a moper is run on the robot controller instead of robot manufacturer provided robot control software. Every sample period the moper collects data from robot peripherals like servo-drives, teach pendant, I/O signals, etc. The data are sent to the ROBOL PC via the parallel connection. Using the data as input, ROBOL generates a new set of data that are sent back to the robot controller via the same connection. The new data are distributed by the moper among the robot peripherals and the robot arm takes a desired action. The parallel communication allows real-time control of the robot arm by a PC that has a different processor and bus architecture than the computer inside the robot controller (LSI 11/73). No major hardware work was necessary.

The serial connection is used to download the moper during start-up, to read messages by the moper, and to send commands to it.

4 ROBOL under RTLinux

ROBOL has been divided into RT threads and a Linux process in the following way:

4.1 RT Threads

The hard real-time part of ROBOL has been divided into three RT threads:

1. *Work-horse RT thread.* It is a good name. The Work-horse is responsible for all work that has to be done during one period and is time critical. For example, communication via parallel connection, monitoring teach pendant buttons, executing commands entered by a user or by a user programme, generating trajectories, preparing commands for robot peripherals, etc. If ROBOL runs in so called dry run mode, i.e. communication with the robot controller is not used, the Work-horse runs as a periodic RT thread. If the communication is active, the Work-horse is interrupt driven by signals from the robot controller.

2. *Auxiliary RT thread.* ROBOL has got two very simple custom made, memory resident

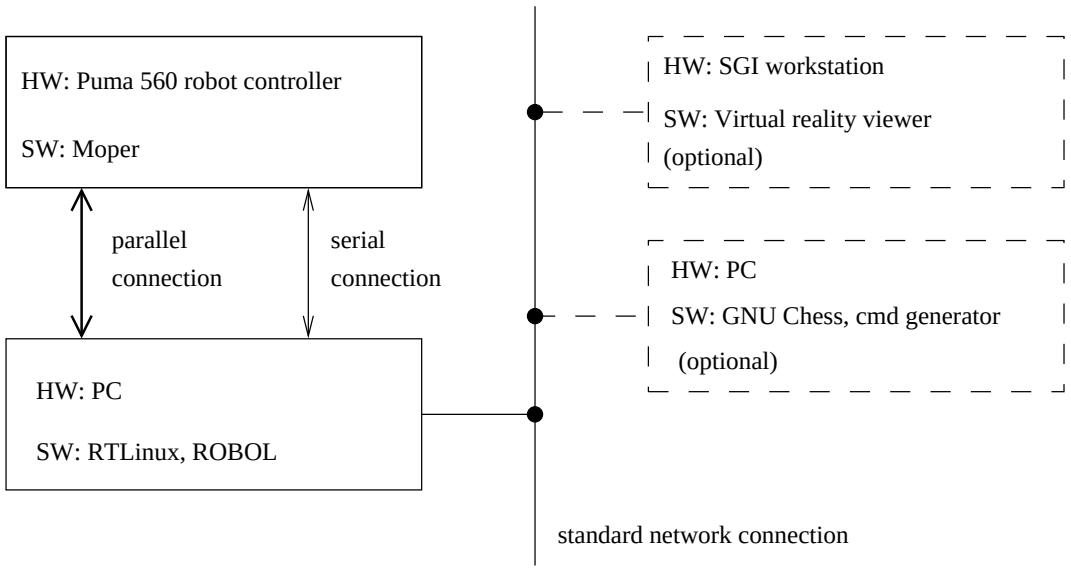

FIGURE 1: *System Architecture*

databases. They are used for storing user programmes and poses. Each database has a set of functions that enter, modify, delete or replace items. The databases and functions are located in the real-time part of ROBOL. The functions are mostly used by the Work-horse because it needs the database items in real-time during user programme execution. The functions also must be called from Linux process part of ROBOL, e.g. during loading/saving the user programmes from/to a disk. As a RT thread can preempt Linux at any time, the databases can get damaged easily. This problem is solved using the Auxiliary RT thread. If the Linux process part of ROBOL needs a database operation, it just starts the Auxiliary RT thread that makes the operation. Shared memory is used to exchange data between the Auxiliary RT thread and the Linux process. As both the Work-horse and Auxiliary RT threads have the same priority, they can never preempt each other so the integrity of the databases is guaranteed.

3. *Monitor RT thread.* It is a periodic, highest priority RT thread. Introducing the Monitor is essential. If a RT thread consumes all CPU time for any reason (well, because of bug occurrence), Linux is permanently preempted. As the Monitor RT thread has higher priority than the other two RT threads, it can preempt them and check how much CPU time they consumed during a period. If things go wrong, an error message is written and all RT threads are suspended.

4.2 Linux Threads

The Linux process part of ROBOL has been divided into four several threads:

1. *User Interface.* This is a highest priority Linux thread because ROBOL must react quickly enough on commands entered by a user. The old MS-DOS ROBOL used an outdated text based user interface. The interface was rewritten using X library (Xlib), the lowest level C language programming interface to the X Window System. Xlib itself does not contain some very low level keyboard functions required by ROBOL, but they were found in [2]. Linux provides plenty of nice tools to develop a modern graphical user interface e.g. [13, 4, 14].

2. *Disk Operations.* This thread is responsible for loading and saving user programmes and other data from/to disk. It is a low priority thread.

3. *Virtual Reality Viewer Communication.* This is an optional thread. The viewer, running on a remote computer, provides a virtual reality (VR) model of the robot workcell and animation of workcell's activity. A user can use ROBOL and the VR viewer for creating and testing user programmes off-line. The animated VR model of the workcell provides him/her the necessary visual feedback. The

basic idea is very simple. ROBOL has its internal robot workcell model containing list of objects, their mutual pose coordinates and relations. Every ROBOL's sample period, the list of objects and objects' pose coordinates are sent to the VR viewer for display. TCP/IP sockets and XDR (eXternal Data Representation) are used for communication. A more detailed description can be found in [3].

4. *Remote Robot Control Communication.* This is an optional thread. Sending commands for immediate execution to ROBOL from a remote computer is interesting for some applications. For example, if a camera is affixed to a robot arm, the camera pose can be controlled directly from a vision system. The remote robot control option is provided as a C function library. It contains functions like opening/closing communication with ROBOL, sending commands to ROBOL, getting current pose of camera, etc. A user links his software with the library.

   Another example application is the robot workcell playing chess. GNU Chess moves are transformed into ROBOL commands. The commands are sent to the ROBOL's PC using a standard network connection and executed by ROBOL. Again, TCP/IP sockets and XDR are used for communication.

5 Implementation Issues

ROBOL was originally implemented under MS-DOS as one task divided into two threads. One thread (containing most of the code) was responsible for all real-time stuff, the other one for all non real-time stuff. A simple thread scheduler was written. The real-time thread could preempt the non real-time one at any time and take as much time as it needed. The only exception were the database functions shared by both treads. If called from the non real-time thread, the functions could block scheduling for very short time. Both threads run in the same memory context.

It was not difficult to divide ROBOL into RT threads and Linux threads, as described in Section 4, because the real-time and non real-time part of it were clearly separated. The user programme and pose databases were the major trouble source. They were rather poorly designed and flooded by pointers and pointers to pointers. The databases have suddenly to be used in two different memory contexts. A given pointer may be correct in one memory context only... Obviously, the correct solution is to write new databases but it has not been done till now.

The real-time part of ROBOL, compiled as a loadable kernel module, needs some functions provided by the standard C library. Some string operations, sprintf and sscanf are required, but not provided by the Linux kernel. Kernel's sprintf does not support floating point number formating essential for ROBOL. Linking a kernel module with the standard C library is possible but it is just a very reliable way to a computer crash. To solve the problem, the necessary source code from glibc was compiled with loadable module gcc options. The result was a light weight library of some selected standard C functions. The library contains plenty of functions that are ,,forbidden in RT threads," but it works well with ROBOL. The library is not provided online and using it is discouraged. In spite of this, several people were given a copy and they do not complain. They need mostly sprintf function supporting floating point number formatting.

Most of the ROBOL code is compiled as a loadable kernel module. Due to huge amount of the code, debugging is of great importance. A special version of ROBOL intended for debugging purposes only is generated together with the real-time one but it is no entirely satisfying solution. Since version 2.3, RTLinux provides capability to debug RT threads using the standard debugging tool gdb. It is really appreciated.

6 Conclusion

ROBOL, the robot control software presented in this paper, is no high end piece of software engineering. However, anybody faced with implementing existing or writing new robot control software for RTLinux will cope with the same or at least very similar problems as the author did.

References

[1] Pavel Andris, Ivo Berka, Karol Dobrovodský, Ladislav Janáč, and Peter Kurdel. Integrated Programming System for Robot Cells Control. In *Proc. of the 1st International Meeting on Robotics in Alpe Adria Region*, pages 154–161, Portorož, Slovenia, June 21-23, 1992.

[2] Amber J. Benson, Gery Aitken, Erik Fortune, Donna Converse, Gregory Sachs, and Will Walker. *The X Keyboard Extension: Library Specification*, 1996. X Consortium Standard.

[3] Karol Dobrovodský, Pavel Andris, and Peter Kurdel. A Virtual Reality Robot Workcell Simulator. In Karel Jezernik, editor, *Proc. of 9th*

International Workshop on Robotics in Alpe-Adria-Danube Region, pages 331–336, Maribor, Slovenia, June 1–3, 2000. Institute of Robotics, Faculty of Electrical Engineering and Computer Science, University of Maribor.

[4] `http://www.gtk.org`.

[5] `http://www.ittc.ukans.edu/kurt/`.

[6] John Lloyd. *RCCL/RCI Hardware Installation Notes*. McGill Research Centre for Intelligent Machines, McGill University, Montréal, Québec, Canada, 1992.

[7] John Lloyd, Mike Parker, and Gaylord Holder. Real Time Control under UNIX for RCCL. In *3rd International Symposium on Robotics and Manufacturing*, pages 237–242, Vancouver, B.C., Canada, July 18-20, 1990.

[8] John Lloyd, Mike Parker, and Rick McClain. Extending the RCCL Programming Environment to Multiple Robots and Processors. In *IEEE Conference on Robotics and Automation*, pages 465–469, Philadelphia, Pa., April 24-29, 1988.

[9] Paolo Mantegazza. DIAPM-RTAI for Linux: WHYs, WHATs and HOWs. In *Proc. Real Time Linux Workshop*, Vienna, Austria, Dec. 13-15, 1999. Institut for Machine and Process Automation, Vienna University of Technology.

[10] Douglas Niehaus. Effective Real-Time System Implemetation with KURT Linux. In *Proc. Real Time Linux Workshop*, Vienna, Austria, Dec. 13-15, 1999. Institut for Machine and Process Automation, Vienna University of Technology.

[11] `http://www.rtai.org`.

[12] `http://www.rtlinux.org`.

[13] `http://www.troll.no`.

[14] Brent B. Welch. *Practical Programming in Tcl and Tk*. Prentice-Hall, second edition, 1997.

[15] Victor Yodaiken and Michael Barabanov. RTLinux Version Two. In *Proc. Real Time Linux Workshop*, Vienna, Austria, Dec. 13-15, 1999. Institut for Machine and Process Automation, Vienna University of Technology.