EUROPEAN ORGANIZATION FOR NUCLEAR RESEARCH

CERN - LEP DIVISION





CERN/LEP-BI/89-64

# USE OF REAL-TIME KERNELS IN PROCESS CONTROL : AN APPLICATION TO BEAM INSTRUMENTATION

D.Brahy, F.Momal, M.Rabany, R.Saban, A.Thys.

Paper presented at the
International Conference on Accelerator and Large Experimental Physics Control Systems
Vancouver - Canada
October 30th - November 3rd, 1989

Prévessin, October 1989

# Use of Real-Time Kernels in Process Control: an Application to Beam Instrumentation

D.Brahy, F.Momal, M.Rabany, R.Saban, A.Thys.
CERN LEP-BI / CS

## ABSTRACT

Process control system components have evolved from a state where they provided a simple digital and analog interface to the world, to very sophisticated dedicated computers offering a service. The software in these devices has followed the same evolution: on the one hand, it has continued satisfying the real-time constraints dictated by process control, on the other hand, it has coped with the complexity of a full blown computer system which it had now become.

The beam instrumentation of LEP consists of several types of instruments which are distributed over the whole perimeter of the collider and are interfaced to the control system using approximately 100 VME crates. Since most of the applications needed the same type of facilities, protection mechanisms and system services, it seemed justified to provide a common environment to all. The embedded applications environment is built around a commercially available operating system kernel and provides services such as alarm management, communications, logbook, etc. The design goals and benefits of this approach and the description of this environment are discussed.

# USE OF REAL-TIME KERNELS IN PROCESS CONTROL : AN APPLICATION TO BEAM INSTRUMENTATION

D.Brahy, F.Momal, M.Rabany, R.Saban, A.Thys.

CERN LEP-BI / CS

## INTRODUCTION

The devices which the Beam Instrumentation Group has to interface to the control system are very different in their nature; they range from electrostatic pick-up based beam orbit measurement system to silicon strip based interaction rate and background monitors, beam current transformers, Q-meter, UV telescopes, a beam synchronous timing system, luminescent screens, split foil monitors, beam stoppers and collimators [1,2].

These devices are connected to the LEP control system using 100 VME based processors which in turn are connected to 25 XENIX-running Olivetti computers. These are networked using TCP/IP over an IBM token ring and are interfaced to the VME crates using an in-house designed special purpose protocol running over MIL/STD-1553 [3]. The application programs handling the instruments run under the supervision of the industry-based operating system kernel RMS68K [4] and take advantage of a run-time environment specially designed to the needs of beam instrumentation. This environment [5] has been the object of a project which span over 6 years. The design goals of the project can be summarized as follows:

- install, adapt and when needed, create program development and debug-

ging tools

- provide a run-time environment adapted to real-time embedded application programs
- provide a simple to use application generation tool (the builder)
- minimize the programming effort by designing a common environment which satisfies the requirements of the different instruments

## THE PROGRAM DEVELOPMENT TOOLS OF THE APPLICATION PROGRAMMER

Following the recommendation of a CERN standardisation working group [6], the beam instrumention group used a set of tools which were partly designed at CERN and maintained by staff in the PRIAM project [6]. One of the goals that the group pursued was to minimize the number of programming languages an therefore the maintenance effort needed for the application programs.

### THE PROGRAM DEVELOPMENT SYSTEM

The working environment of the application programmers coonsists of a local area VAX cluster running VMS which contains the language processors and the builder. The application tasks are down-line loaded into the target processors either using the serial line or via the MIL/STD-1553. The debugger, when activated, runs in the target system.

### THE LANGUAGE PROCESSORS

There were already available at CERN a set of programming languages (FORTRAN, Pascal and the assembler) for the 68000 family processors. Language mixing was possible. It was decided to use

Pascal on the ground that the restrictions imposed by the language would ease program writing and debugging for non software specialists. Where fast code was needed, the 68000 assembler was used and interfaced to Pascal [7, 8, 9, 10].

## THE BUILDER

In order to enforce security against system and applications corruption, the CPU module was required to have memory management. The application programmer designs his system as a set of independent execution units called tasks. The builder [11] allows to specify for each task the memory segments needed, as well as the common libraries and the object modules making the task code. It is organized

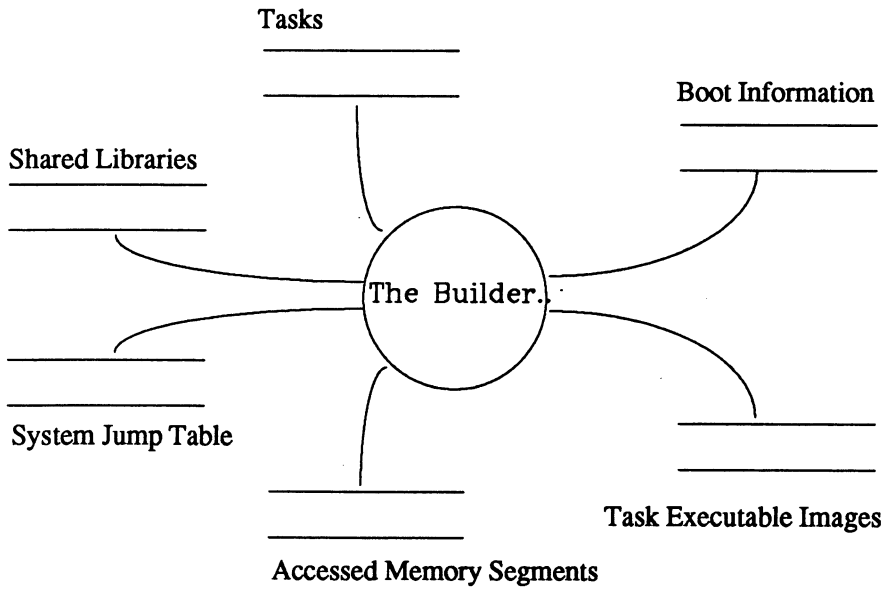


Figure 1 : The Builder Data Flow Diagram

as a pop-up menu system which maintains an information file for each application and generates the whole application on request by compiling or assembling the source files, linking them to the libraries and to the system services jump table.

## THE RUN-TIME DEBUGGER

An in-house designed run-time debugger which allowed the debugging of a single task at a time was used. Together with the classical functions of inspection registers, memory locations and breakpoints, this debugger allowed system parameters to be inspected and changed. Tasks could be activated or stopped, events and interrupts generated [12, 13].

## THE RUN-TIME ENVIRONMENT OF THE EMBEDDED APPLICATION SOFTWARE

A study of the facilities needed by embedded applications programmers was made by a team of software specialists and system integrators. A number of areas where support was needed were identified and the strategy of implementing a common set of facilities for the different devices was chosen.The facilities were implemented as a set of libraries and, where needed, as a set of servers which are accessed by an interface library.

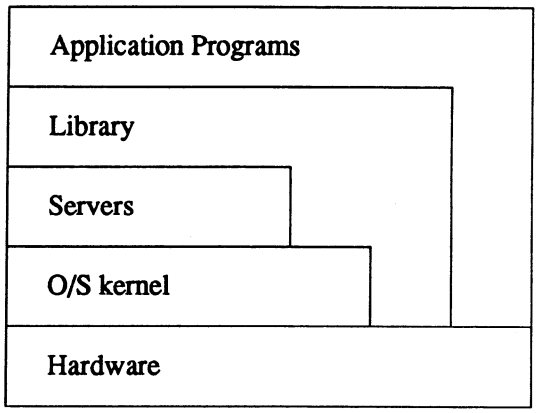


Figure 2 : Embedded Software Layering

The structure of both the libraries and the servers have followed the same blue print, thus easing the writing and the maintenance of the code for the different facilities. All was coded in Pascal [4, 14, 15].

## ALARM SERVICE

The LEP Control system supports an alarm processing service using a central database to inform the

operator of the action to perform upon the arrival of an alarm, its gravity, an explanation and the details of the on-call service. Each device can initiate an alarm message indicating a fault state or a fugitive abnormal situation.

The alarm service is a set of library calls which enable the application programmer to make use of the LEP alarm system: the application programmer informs the system of the alarm and leaves the follow-up to the system. A mechanism which filters the alarms prevents high repetition rates and repeats the alarms until an ackowledgment from either a higher level computer or from a human operator (when needed) is received. An active alarm list is kept in each device: it allows the verification that the status reflected in the control center is correct and it is used when one of the computers dealing with the alarms restarts and needs to be briefed on the status of the alarms.

## BEAM SYNCHRONOUS TIMING

The BST [16] system delivers to each of the beam instrumentation station a timing pulse every 88.9 us which is precise within 50 ns. Along with this pulse, 7 bytes control the behaviour of the instruments. An additional byte is used as return information. Some of the timing signals are directly delivered to the hardware and used as acquisition strobes, while others are delivered to the sofware in the form of two fields: code and data. The former is used to trigger a particular program while the latter provides data to the program itself. A set of library calls allows application programs to wait for a particular code. The latency is of the order of a few milliseconds.

## COMMUNICATIONS

Presently, the LEP control system provides a communication system which uses the MIL/STD-1553 [ 17, 18] as its physical medium. The in-house designed protocol allows transactions between a bus master and any of up to 31 bus slaves. Specially designed for equipment control, this protocol is mainly used in 'command-response' mode where a program residing in the parent computer issues a command which conveys data in either direction. A generalization of this mechanism permits the slave-initiated transaction of the 'command-response' type.

A set of library calls permits embedded application programs to provide a service to the users of the device by entering a 'wait-for-command' mode or ship data acquired following a beam synchronous timing command to the parent computer using the 'slave initiated command-response mode'. Up to 32 kbyte of data can be shipped in either direction in a single transaction. The data transfer rate has been measured to be 16 kbyte/s.

## EXCEPTION MONITOR

A special task in the system keeps track of all unwanted exception situations which might be caused by a running application program ( bus error, zero divide, address error, etc). A set of primitives allows the embedded application program to describe the processing desired when each exception condition occurs: a task can be aborted, restarted or continued after the exception. An alarm or a log book entry can be generated. With the help of special tools the specialist can monitor the behaviour of a particular device.

## FILE SYSTEM

The simple file system extends the XENIX file system to the embedded application programs. Tasks running in the CPU installed in the VME crate can access files situated in the file system of the parent computer. Essential primitives such as open file, close file, delete file, verify the existence of a file, write block and read block have been implemented. They use the XENIX path name for file naming and comply to the XENIX file protection rules.

## GENERAL MACHINE TIMING

The LEP control system allows the broadcasting of events which are used to drive the collider through a predefined sequence. This is achieved by informing the different components of particular happenings called events [19].

Embedded applications programs can suspend their execution until the desired event has occurred. They can also instruct receiver hardware to generate a pulse on the occurence of an event. The general machine timing clock ticks every millisecond and apart from machine events, conveys the date and the time to the devices in the field. Time information is used to stamp the acquired data.

## LOG BOOK

Each device has at its disposal a circular stack which can hold at most 32 messages. The logbook primitive allows the introduction of a character string which is 40 characters long and which is kept in the stack together with the date of its insertion and the name of the program which has generated it.

Events which need not be sent to the alarm system but which represent useful information when an abnormal situation is encountered, are logged by means of this facility.

## NON-VOLATILE DATA MANAGER

Calibration data, device names and device data needs to be stored and to be always available even after power loss. The data manager provides simple primitives to create a data block, read, write and delete it. This data is kept in a battery back-up memory residing in the CPU board. Protection rules allow the utilization of the data by several programs in a secure manner. This facility provides a relatively fast access (1-2 ms) to stored data of up to 32 kbytes.

## CONCLUSIONS

From the experience gained in the first three months of operation, the system can be considered to be running satisfactorily. Some careful tuning of relative task priorities will improve the performance significantly. Where the interrupt latency of the kernel (200 us) is not acceptable, the interrupt is treated outside the kernel thus reducing the response to the minimum. This practice has not shown any disruption to the operation of the system. The only real problem encountered is the corruption of the same particular area of battery backed-up memory during the occurence of a bus error. This event happens with a frequency just above once a day over 100 systems. An automatic recovery procedure hides this from the operation; nevertheless a serious investigative effort is being made to extract the bug from the system.

In the partricular context of the beam instrumentation group where in most cases the programmers of

embedded applications were the designers of the interface hardware and not trained programmers, the use of a support team manned by system integration and software specialists was the only practical means of providing the necessary tools and services.

Although the application generation cycle remains still to be reduced, the builder has proved to be an essential tool for the production of embedded software. The language processors produce satifactory code, but in some cases they are deficient since they were not designed from the beginning to be used in a real-time multi-tasking environment. The debugger is indispensable for embedded software, but again it should from the beginning be designed as a tool to run in real-time multi-tasking environment.

Considering the relatively high cost of the project ( about 7 man years ), the development of a common set of facilities ( about 35000 lines of Pascal, C, assembler and make files ) for as many as 10 different instruments is no doubt an economy in man power.

The use of the tools by a relatively large population of application programmers (6)has permitted a faster debugging cycle by distributing the load.

## REFERENCES


[ 1] Beam Diagnostics for LEP, C.Bovet, 13th International Conference on High Energy Accelerators, Novosibirsk, USSR 7-11 August 1986

[ 2] MIL1553 Interface to the Beam Instrumentation of LEP. LEP-BI and SPS/LEP-ACC

[ 3] LEP Controls System: Architecture, Features and Performance, PG.Innocenti. SPS/89-35 (ACC)

[ 4] M68000 Family Real-Time Multitasking Software User's Manual - MOTOROLA MicroSystems

[ 5] Environnement de Support pour les logiciels d'application des ECAs de LEP-BI, D.Brahy, F.Momal, T.Pauwels, R.Saban, A.Thys, LEP-BI Note 88-1

[ 6] Computing at CERN in the LEP Era - May 1983 Editor: Erwin Gabathuler

[ 7] M68MIL CROSS MACRO ASSEMBLER, Horst von Eicken, CERN Yellow Report 83-12

[ 8] CERN CONVENTION FOR PROGRAMMING THE MC68000 FAMILY, edited R.Cailliau, CERN Yellow Report 84-12

[ 9] User Guide to the Pascal Cross Compiler for the Motorola 68000, D.Foster, CERN DD.

[10] Use of Microprocessor Cross Software under VAX VMS, JD.Blake, CERN/DD/SW-LM/T9

[11] APPLMNG Application Manager, Le manuel de l'utilisateur, F.Momal, R.Saban, LEP-BI Note 89-2

[12] MoniCa Symbolic Debugger, Horst von Eicken, CERN DD 86

[13] RMON user's manual, D.Kemp, CERN-DD August87.

[14] A Pascal interface to RMS68K - User's Guide, D.Brahy, F.Hemmer, R.Saban, LEP-BI Note 87-6

[15] The LEP/BI RMS68K based development system, D.Brahy, F.Hemmer, R.Saban, LEP-BI Note 87-3

[16] The Beam Synchronous Timing for the LEP beam instrumentation, G.Baribaud, D.Brahy, A.Cojan, F.Momal, M.Rabany, R.Saban, JC.Wolles, this conference.

[17] Protocols for the LEP/SPS Control System, J.Altaber, PS.Anderssen, K.Kostro, D.Lord, SPS/ACC Note 86-8

[18] The LEP/SPS access to equipment, LEP Controls Note 54

[19] TG3 -V Hardware Description, G.Beetham, B.Puccio, November 1986.