DLX, The Multiprocessor Assembly for LEP/SPS Controls

P.D.V. van der Stok
European Organization for Nuclear Research
CERN 1211 GENEVA 23, Switzerland

Abstract

The computers composing the controls network for LEP are VME based assemblies of Motorola 68000 microprocessors. A complete VME crate with microprocessors is called DLX. The operating system (O/S), ELECTRE, running on the processors is based on a real-time executive called SCEPTRE, which has been standardized by the french BNI.

Two versions of the operating system exist: a multiprocessor and a monoprocessor version. For LEP/SPS controls the monoprocessor version is used to add intelligent controllers, token-ring, MIL-1553-B, to the multiprocessor version; while the multiprocessor version is used for the execution of real-time control Tasks. The main components of SCEPTRE, ELECTRE and the additions required for LEP controls will be presented.

Introduction

In 1982 the decision was taken that the LEP controls system and the SPS controls system should be merged and that the control should be executed from one control room [1]. An analysis of the SPS control system in that period showed that very little interaction existed between the individual control processes and that many of them could be executed in parallel as they did not share any resources. The idea was put forward to replace the then existing N100 computers by multimicro assemblies, a less costly and more flexible solution than the installation of a mini computer[2]. In the assembly only standard reusable modules will be used to configure an assembly to the required needs. Also later modifications of the assembly requirements can more easily be integrated than would be possible with a manufacturer controlled mini computer.

A first VME based prototype was contructed at CERN in 1984. With the thus required experience a call for tender was issued, which resulted in the acquisition of the DLX system constructed by THOMSON-SINTRA [3].

Additions have been specified for the DLX to make it conform with the specifications of the controls and especially to make it compatible with the SPS control system [4],[5],[6].

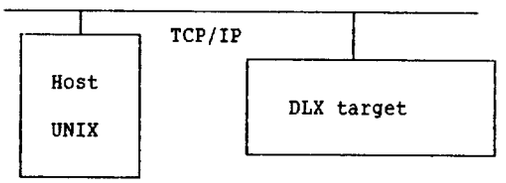
Software development

However good your computer system, when the development, modifications and testing of your control procedures are prohibitively difficult more complaints than enthusiasm will be generated. Both the development requirements and the control requirements influence each other and they are in turn influenced by the DLX structure. A description of the software development environment gives consequently a first glimpse of the DLX system and an idea of the place of the DLX inside LEP/SPS controls. A first discussion of the development cycle will certainly satisfy the people who think that the usefulness of computer system is completely determined by its development environment.

For the development of DLX software two stages can be discerned:

- The contruction of the complete ELECTRE O/S plus additional fixed applications.
- The incremental addition of applications during the lifetime of the generated ELECTRE O/S in a particular DLX.

The DLX is explicitly intended for the execution of the control procedures. The actual development of the control procedures will be executed on a more appropriate UNIX based host machine and consequently all development is based on the host target concept.



The host is a VAX, a ND500 or an equivalent UNIX based host system, while the target is a complete DLX assembly or one CPU in a DLX assembly. The network connection is UDP and TCP oriented and the physical medium for the DLX is token ring.

For the cross development, compilers (C, FORTRAN-77 and MODULA-2) are available to compile the individual files which constitute the applications and the O/S. The generated object format is COFF 5.2 which contains the 68000 code and the debug information. The generation of ELECTRE is done on the host. ELECTRE is completely written in the C language with a very small machine dependent part written in assembler. Its individual files are compiled separately on the host and with a linker/loader they are merged to a set of packages. A configurator utility will, based on a configuration file, load all the required packages into as many object files as there are CPUs in a particular DLX assembly. The latter objects are either loaded into Eprom or are directly loaded into the RAM of the individual CPUs.

The debugging is done with the aid of the symbolic cross-debugger: SPY. Again it is host target based and most of the DLX machine is visible to SPY. Its main tasks are:

- (repetitive) visualisations of all Task descriptions.
- (repetitive) visualisation of existing Tasks and their state.
- (repetitive) visualisation of the values of variables.

Breakpoints can be set and the state of the total machine can then be inspected. It should be noted that a breakpoint freezes the state of all CPUs which constitute the multi.

A second utility "dbx68" is more application oriented. It allows the debugging of one process at the time while all other processes inside the

assembly continue executing. The man machine interface is menu oriented with displays of lists of variables, with zoom facilities to visualise the RECORD, struct or array contents. Also it is possible to follow pointer structures to inspect trees of RECORDS. Dbx68 knows about the syntax of the three supported languages (must be explicitly declared by user), while the semantics is taken from the COFF 5.2 debug information.

Once a complete DLX is configured, its fixed applications and the operating system can be debugged with SPY. Additionally it is possible to incrementally add applications which can be debugged with dbx68, without interference with the other already existing Tasks.

DLX hardware lay-out

A short introduction of the used DLX hardware will be presented here. The DLX consists of a VME crate which contains a certain number of "UTM10" cards. They contain a MC68010 microprocessor, double port memory visible form both the VME and the CPU, local memory only visible from the CPU, a MMU, and two serial interfaces. The ELECTRE multi O/S can run on a number of one till n CPUs. Standard VME cards can be added under control of the multi O/S. One CPU on the multi will execute the driver for a specific card. Intelligent cards, based on the UTM10, can be added under the control of the multi. They may contain a mono version (no MMU) of ELECTRE and communicate with the multi through a communication package. In the case of the LEP/SPS controls intelligent controllers of this kind are foreseen for the token ring and the MIL1553b connections [7],[8],[9].

ELECTRE basics

ELECTRE is based on a kernel (SCEPTRE), which is defined by the french Bureau d'orientation de la Normalisation en Informatique (BNI). The kernel contains a set of basic primitives on which a more elaborate operating system can be built. The basic primitives are in short:

- Wait for event or send event to specified Task.
- Enter or extract an element to/from a FIFO.
- Enter or leave a protected Region.
- Start, stop or continue a Task.

In ELECTRE these primitives execute in the multi version or the mono version in exactly the same way. Once created the user does not need to know on which CPU a Task is situated, neither does he need to know on which memory the kernel objects are located.

To allow a consistent and safe addition of facilities on top of the kernel the concepts AGENCE and DOMAIN are introduced. An AGENCE is a package which contains a certain number of facilities (typically a set of PROCEDURE calls). Inside the AGENCE a number of objects local to the AGENCE are defined. Instances of these objects are created inside the AGENCE and completely under the control of the AGENCE. An AGENCE is allowed to invoke the utilities of other AGENCEs or of the kernel.

The DOMAIN defines the area which can be accessed by a Task at a given moment during its execution. In other words: when a Task has entered a Domain, the code area and the data area of that Domain is only accessible to the Task at that moment. One or more AGENCEs can be grouped in a DOMAIN. All existing AGENCEs (which together constitute the ELECTRE O/S) can be groupd in one DOMAIN, while new AGENCEs, which are not bug free, can be put in different DOMAINs. Consequently the O/S is potected againts the errors of the newly created O/S features.

The distribution of the AGENCEs over DOMAINS and the distribution of DOMAINs over the CPUs of the multi are specified in the configuration file. The latter file is used by the configure utility to create the object files for the individuals CPUs.

The mono version of ELECTRE does not know about DOMAINS (no MMU).

The main AGENCEs which contitute ELECTRE will be enumerated and explained below. N.B All AGENCEs use the facilities of the kernel mentioned above.

AGENCE of Tasks. This AGENCE provides the link between the kernel primitives which act on pointers and the Task primitives which act on Task names. Additionally the dynamic creation and elimination of Tasks is done here.

AGENCE of clocks. This AGENCE provides the facilities needed for all timing needs of the Tasks. Tasks can wait for delays or initialise timers. The timers will send signals to Tasks when their specified time period has passed.

AGENCE of semaphores. This AGENCE provides semaphore facilities which can be used over Domain boundaries. In the C library additional semaphore facilities are provided for semaphores which are used locally in one Domain. They complement the Region facility of the kernel, which may have undesirable side effects if not used with caution.

AGENCE of mailboxes. This AGENCE may create and suppress mailboxes which can be used over Domain boundaries. They complement the FIFO concept of the kernel which can only be used locally inside a Domain.

AGENCE of the I/O system. This AGENCE provides a homogeneous interface to all I/O from ELECTRE. It receives blocks of data and prepares blocks of data from/to the user programs. The data are picked up by the specific device drivers and passed from or to the device. In this way data can be driven to any Task independent of its CPU to any device independent of the CPU to which the device is physically attached.

AGENCE of the file system (SGF). This AGENCE contains a large part of the UNIX 5.2 file system. Not all facilities are included to guarantee response times, and to minimise the code size. Drivers for RAM, floppy disk and winchester based files are provided.

AGENCE of exceptions. This AGENCE allows the continuation or controlled stop of control Tasks in the event of an exception. Tasks can be activated on a set of specified exceptions provoked by a specific Task. After treatment the provoking Task can be aborted or its execution can be continued. The exception causes are memorized in memory and can effaced at will by the treating Tasks.

LEP/SPS control additions.

The major task of the DLX is to control equipment. The connections between the DLX and the equipment

are implemented via MIL1553B. The instructions from the main control room or from other consoles to the DLX arrive over the token ring network. The protocol on the network is TCP/IP and UDP/IP. On top of UDP/IP a special file access and remote job access protocol (RATP) is implemented [4]. Per DLX a set of 1 till 8 UTM10s are foreseen for the equipment access and 1 UTM10 for the token ring access.

The tasks to be executed by the DLX are manyfold:

- Execute autonomous surveillance programs.
- Receive and execute Remote Procedure Calls.
- Receive and execute NODAL IMEX and EXEC calls (SPS compatibility).
- Send RPC, IMEX and EXEC commands to other DLXs.
- Send equipment access requests to correct equipment interfaces.

All requests to the equipment are symbolic. The Equipment Directory Unit (EDU) binds these requests to the correct physical addresses on the MIL1553B and dispatches them. The EDU guarantees that only one transaction at the time is going on with one particular piece of equipment.

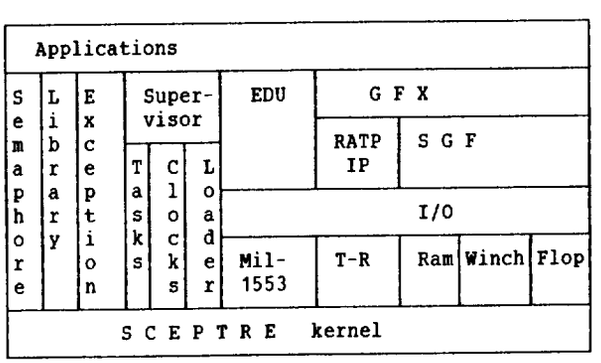
Via the token ring two types of action can be done:

- Request remote job execution, execute remote request.
- Ask for remote file access, execute remote file request.

Further local file and device access can be done through the file system (SGF). The AGENCE GFX provides one OPEN call for remote job -, remote file -, local file -, or local device access. GFX will determine which package has to be invoked for any particular request. Additionally it will be possible to access the AGENCE of the mailboxes through GFX to provide "pipe" like accesses to other Tasks.

Another addition required for the LEP/SPS closely linked with the above requirements is the possibility of dynamically loading and executing Tasks. The DLX system was originally only foreseen to execute fixed applications. An AGENCE for dynamic loading has consequently been added.

A strict organization of the Tasks in the DLX is needed to estimate its load at a given time and to determine where new applications can be added. In connection with the dynamic loader the Supervisor AGENCE was introduced. The latter recognises different types of Programs and Tasks. In our nomenclature a Task executes a Program.



The following combinations of Tasks and Programs are handled by the Supervisor:

- Permanenlty loaded programs.
- NODAL interpreter programs resident on file.
- Dynamically loaded programs from a file.
- Repetitively executing Tasks.
- Tasks started by accelerator events.
- Tasks started on a clocks tick.
- Tasks started by a remote request (RPC, EXEC, IMEX).

Below a diagram of the ELECTRE software layout including all AGENCEs discussed above is shown.

Conclusion.

The structure of the DLX multiprocessor assembly has been presented. Its place inside the LEP/SPS controls system is explained. In the immediate future an analysis and tuning of the DLX behaviour will be needed to obtain the maximum benefit from the multiprocessor capabilities.

References.

[1] M.C. Crowley-Milling, The control system for LEP, 1983 Particle Accelerator Conference, 21-23 march, 1983, Santa-Fe, new-Mexico.

[2] J.Altaber, M.C. Crowley-Milling, P.G. Innocenti, R. Rausch, Replacing Mini-Computers by Multi-processors for the LEP Control System, 1983 Particle Accelerator Conference, 21-23 march,Santa-Fe, New-Mexico.

[3] P.D.V. van der Stok, Notions sur le systeme multiprocesseur DLX et son executif temps-reel ELECTRE, LEP Controls Note 77.

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

[5] J. Altaber, P.D.V. van der Stok, DLX Agences for the LEP/SPS Control System, SPS/ACC/Note 86-9.

[6] J.Altaber, A. Bland, P. Brummer, V.Frammery, The LEP/SPS Access to Equipment from DLX, SPS/ACC/Note 86-22.

[7] A. Bland, The Communication package for the equipment network of the LEP and the SPS Accelerators. This conference.

[8] R. Rausch, Real-Time control networks for the LEP and SPS Accelerators, this conference.

[9] P.S. Anderssen, The new control room infrastructure for the LEP and SPS Accelerators. This conference.