Real-Time Systems For Local Intelligence

T.T. Luong

GANIL, Caen, France

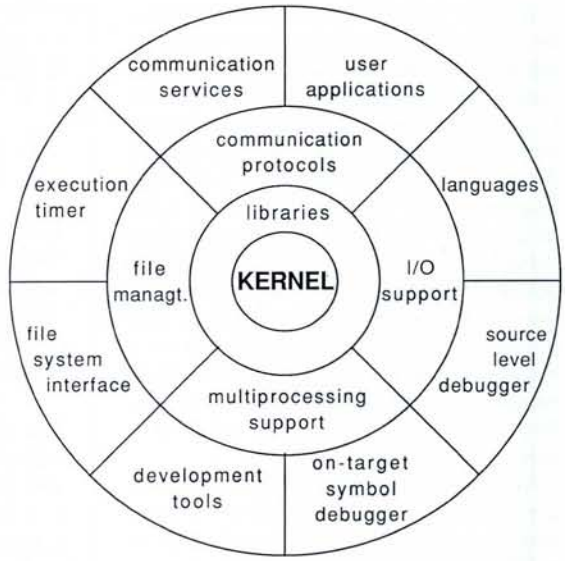


Fig. 2 — The structure of an operating system for a multiprocessing real-time process control system. A small, high speed kernel providing essential services for real-time applications lies at the core.

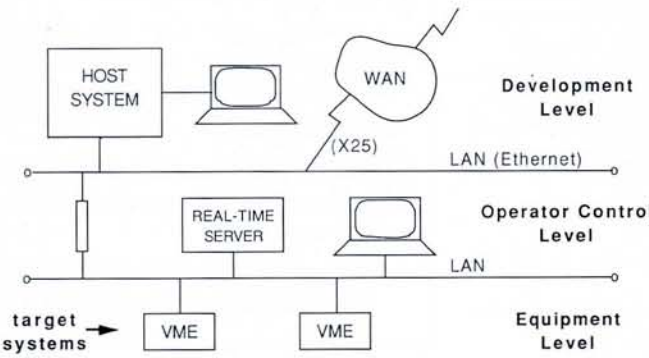
The increasing complexity of equipment in industry, automation, image processing and experimental physics demands more and more sophisticated facilities for control in real time.

A real-time control system is an information processing arrangement that keeps track of critical, external, real world events with sufficient speed and reliability to ensure the correct response to stimuli within a finite time. Computer-based real-time systems are essential for controlling processes in the real world environment because life's everyday phenomena are generally far from digital in character and they undergo nonlinear (*e.g.* temperature changes) or unpredictable (*e.g.* equipment failures) variations over a large range of time periods extending from microseconds to hours or more.

Inserting local intelligence at the front end of devices is the key. The tremendous progress in microprocessor technology combined with plummeting costs means that conditions are ripe for doing this. The control of items of equipment can then be achieved in an incremental and modular fashion by introducing recent advances in networking technologies and standardization which allow processors to be federated into flat, democratically distributed, simple architectures.

One could be dazzled by this exciting panorama when choosing a real-time system to cope with the escalating demands for improved capabilities, higher performance and greater cost effectiveness. I shall therefore review the main issues that should be considered when evaluating a real-time system for local intelligence in process control networks.

Fig. 1 — Layout of a multiprocessor real-time system for local intelligence in a process control network connecting host computers and man — machine interfaces to front-end processors (target systems) in VME crates A real-time server is implemented for dedicated real-time interaction with the processors.



Multiprocessing

Real-time systems systems are composed of concurrent *tasks* each dedicated to a specific mission, *e.g.* controlling the polarization current of a magnet, moving a stepping motor, reacting to an alarm signal. The tasks are arranged into *multitasking functional units* which are usually built up into a *multiprocessing* real-time system mapped over several processors in order to share the workload. Fig. 1 shows the layout of a typical multiprocessing real-time control system that was adopted by a physics laboratory to control a heavy ion accelerator.

Communication is the key issue determining the effectiveness of multiprocessing. It is established between processors using buses (*e.g.* field buses, VME — see page 40), or local area networks such as Ethernet. Communication between different functional units employs various techniques (*e.g.* message passing, remote entry calls, remote procedure calls); communication between tasks within a functional unit is achieved using a shared memory or mailboxes.

The *synchronization* of tasks can be performed using semaphores or mailboxes. A semaphore can be viewed as a flag that protects a portion of a program, called the critical section, which manages access to a resource that is not shared, for instance writing on a disk. Tasks waiting for semaphores or mailboxes are in "suspense" so they do not consume central processing time until their reactivation by the arrival of the appropriate message. The ADA programming language offers this so-called rendezvous facility at the language level.

Operating System

The operation of a real-time system requires a high-performance *operating system* which provides:

- real-time multitasking, *e.g.* maintaining task states, interrupts and scheduling in pre-emptive priority or time-slicing mode to trigger a task into action before its deadline;
- intertask communication and communication with the outside world;
- memory management and protection against illegal access;
- error handling with status information.

A real-time *kernel* is the part of an operating system that provides the indispensable services for real-time application, such as task scheduling, fast response to incidental events, interrupt handling. Fig. 2 shows a typical representation of a kernel as the most internal component of an operating system. Since it must be small for high-speed execution, and be endowed with essential capabilities which generally exclude safety mechanisms, careful debugging of the application program is strongly advised.

Languages

An important characteristic of a real-time language is that communication and synchronization are provided by the language itself. Four real-time languages dominate: C, Pascal, Modula-2 and ADA but incomplete standardization effectively means that Pascal and Modula-2 are not prime candidates for real-time systems. C can be considered now as replacing

Than Tam Luon is presently the Controls Group Leader at GANIL, Caen, France. He studied at the Ecole Nationale Supérieure des Télécommunications, Paris.

assembly language. OCCAM, a real-time programming language specifically designed for transputer architectures should also be mentioned.

ADA is unique because it supports structured programming and some of the latest methodologies such as object-oriented design. This language also contains special constructs for embedded system programming (e.g. parallel task exception handling, intertask communication, rendezvous task synchronization, *etc.*).

External Communications

A real-time system communicates with the outside world via three types of services: input/output (I/O), networking and the man-machine interface.

Input/output services of specialized types must generally be added to extend the scope of the standard I/O support provided by the real-time operating system. These modifications can cope with specific requirements of the application program by, for example, permitting specific device drivers to be imported.

Networking allows communication with other systems along local area networks (LAN) or wide area networks (WAN) and the activation of important remote operations such as booting, diskless use, virtual terminal, file access, *etc.* Open, standardized protocols for communication such as OSI of the International Standards Organization must be adopted to ensure easy evolution and implementation and independence from a supplier.

The *man — machine* interface permits, *via* proper console displays, interaction at a high level with the real-time system. One can therefore readily perform operations like program development which require the down-loading of real-time programs into a front-end processor from a host computer for remote debugging or execution of a process. Graphics support and the display of several windows on a console display employing standards such as X Window may be helpful.

Software Development Environment

Cross-development is commonly used for distributed system software. It consists in creating application code on a computerized system called the *host* system and executing this code on a *target* system located in a different computer. This approach is adopted because the target system has been slimmed down to the essentials to maximize performance when running the application. The host environment, meanwhile, is better equipped with powerful tools and various utilities for preparing the application code.

Within the host environment, it is desirable that:

– productivity be sought as far as possible during system development;

– the host and cross compiler derive from the same basic technology (or *root*) to avoid discrepancies between the program codes they generate. An additional advantage is that the user need only learn one type of man — machine interface;

– a *remote debugger* for source code capable of reaching the target through the network should be available. Combined with multiwindowing information display facilities, this will ease the control of program execution and the isolation and tracing of software errors;

– facilities to *simulate* the target environment be provided to ease the testing of embedded programs, and thus to be independent of target hardware such as timing and I/O devices.

Software Engineering

The main objectives of software engineering are significant improvements in software quality and productivity with reductions in the cost of maintaining and modifying programs. They can be achieved using different methods including structured or object-oriented approaches which are invoked by computer-aided software engineering (CASE) tools. A wide variety of CASE tools, based on workstations or personal computers, are now available to cover the major phases of the software life-cycle including analysis, design, coding and testing as well as information modelling and documentation. Real-time features such as control flow graphs, state transition diagrams, process activation tables and state-event matrices expand CASE tools capabilities to support interactive and concurrent processes.

Standards

A key concern is the minimization of investment over the life of a system while improving its reliability. One possible approach is to make use of acknowledged industry *(de facto)* or international open standards to allow: an industrial approach to software development; time saving and reliability; portability (*i.e.* being independent of the hardware); maintainability; the sharing of experience and products (*e.g.* collaboration between laboratories and industrial plants). An example of an international joint effort to propose standards for real-time software is the Open Real-Time Kernel Interface Definition (ORKID) project uniting industry and research.

Conclusions

Implementations of real-time systems for local intelligence are generally at variance with the technical requirements discussed above. This has shown up in a study of the types of real-time systems that are being used by institutes belonging to the EPCS Group. There clearly exists a wide variety of systems that mix industrial and in-house developments. This very undesirable situation persists because implementations depend on local constraints. It can be improved by the "agora" activities of organizations such as the EPCS Group where common issues of importance can be discussed to trigger solutions along some principal directions.

FURTHER READING

Proc. Europhysics Conf. on Control Systems for Experimental Physics, *CERN Report 90-08*, Ed. B. Kuiper (1990).

Proc. of the Workshop on Real-Time Systems for Microprocessors, Chamonix, France, 1989.

Proc. Int. Conf. on Accelerator and Large Experimental Physics Control Systems, *Nucl. Inst. and Methods* **A293** (1990).

Databases In Physics Control Systems

D.P. Gurd

TRIUMF, Vancouver, Canada

One or more databases lie at the heart of modern computer-based control systems used in large-scale experiments. While ensuring that all the computers and subsystems reference consistent data, they also enforce a straightforward discipline for making changes and provide a reliable source of system documentation. I shall describe these databases and discuss the critical issues and compromises facing their designers.

David Gurd recently joined the Controls Group, SSC Laboratory, TX, USA as its Leader having spent 20 years at TRIUMF in Canada. He received his Ph.D. in nuclear physics from the University of Alberta, Canada.

Integrated Technical Database

A complete project description in the form of an integrated technical database should underlie all phases of an accelerator or other large experimental physics project. This database is used for design, simulation, cost estimating, procurement, construction, and eventually for operation and management. The approach was used for CERN's Large Electron Positron Accelerator (LEP) and will be adopted at the Superconducting Super Collider (SSC) in Texas, USA and at the KAON accelerator planned for Vancouver, Canada.

Specialized databases including run-time databases required for accelerator control are extracted from the integrated