CALAS70 – a real-time operating system based on pseudoprocessors

G. HEPKE
Kernforschungszentrum Karlsruhe, W. Germany

1. Introduction

The changing prices of minicomputers mark one direction in which real-time systems will develop: cheap computers will be used in a decentralised manner to solve individual problems. This solution covers quite a number of applications for real-time systems; other applications, however, call for new central systems, to be developed satisfactorily in the future. The concept and design of central systems, however, may profit from the decentralised solution. The processes* to be controlled and computed by the system presented here are long-term scientific experiments. The creative man carrying out the experiments requires a high degree of convenience, and that means a time-sharing system.

2. Principles of system design

The system design is based mainly on:

1. Specific requirements of experimental operation.
2. Advantages of decentralised systems.
3. Precise knowledge of the task profile.
4. Facilitation of implementation, testing, and assembling of system elements.

The specific requirements of experimental operation include, for example, rapid data acquisition and transfer to tape and/or disk, long-term data management, and programs for interactive work with graphic displays.

The advantages of decentralised systems are simplicity, easy handling and programming, reliability of system software, and independence.

A well known task profile allows time and storage saving restrictions by dedicated programs. The system architecture reflects the natural static and dynamic modularity of the tasks of the system and thus facilitates the management of the system.

*i.e. technical processes in this context.

3. Pseudo-processors as system modules

The elements of the system are pseudo-processors which can be considered to be independent dedicated minicomputers. The operating system can be taken as a network of pseudo-processors which are coupled via common core storage; cooperation takes place through special commands.

Each pseudo-processor is represented by a set of registers, a task queue and a program with variables for processing these tasks. As a rule, each task is characterised by a small number of parameters for setting the respective pseudo-processor, and that means programming the program.

By using only one processor and a restricted core memory, the independent pseudo-processors become dependent on each other with respect to time.

The multiplexing of the processor as well as the adaptive processing of the queues are subjects of Mr Herbstreith's paper [1] (see p. 84 of this book).

4. Peripheral devices and pseudo-processors

All activities of the system are integrated into the concept of the pseudo-processors. System and experiment periphery are embedded in pseudo-processors in order to be capable of general cooperation. Thus, the task for a peripheral device is no longer different from tasks for other pseudo-processors; the peripheral device is directly accessible to the user who need not tackle its specifications as check and control bits. Nevertheless, he is directly connected to the hardware of the device, and is able to state as task parameter the interrupt subroutine programmed on its own.

5. States of pseudo-processors and system

In order to increase the transparency and to decrease reaction time and failure rate of the

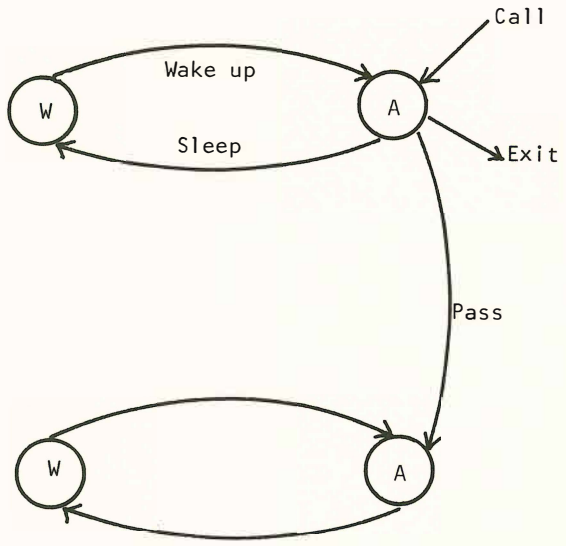

Fig. 1 States of the pseudo-processors

system, the number of relevant states and changes of states must be kept as small as possible. The same is true for commands acting on these states.

The relevant states of the pseudo-processors (see Fig. 1) are:

1. Waiting, for activation through a task or reactivation of a task already started.
2. Active, i.e., able to work or working.

The relevant changes of state for the integral system are:

1. Upon definitive or provisional termination of a task the next active task must be searched for processing.
2. Upon activation and reactivation, respectively, of a more urgent task than the task being processed, the more urgent task is started at once.

6. Communication, control commands

The pseudo-processors are independent units executing processes of specific types defined by tasks. Since, in general, a job includes the sequence of different processes, commands must be available for cooperation between processes and for sequence control.

By CALL a process gives a task to another pseudo-processor. Significant parameters of this command are the name of the pseudo-processor addressed and a task identification which distinguishes up to 23 tasks given by only one single process.

By SLEEP a process changes over to a preliminary state of rest ('waiting') in order to synchronise with other processes initiated by it. Parameters of this command are the task identifications of the corresponding tasks.

By WAKEUP the possibly 'sleeping' initiator of the waking-up process is woken up (set 'active').

By EXIT a process comes to an end and thus the task clears the pseudo-processor. EXIT may imply an automatic SLEEP or an automatic WAKEUP. This mechanism, which is installed as a safety precaution, may block a whole line of pseudo-processors. This blocking can be avoided by use of the PASS command.

By PASS a process transfers a task and is brought to an end unconditionally. This is possible since the 'obligation to wake up' is transferred to the new task.

7. Prevention of deadlocks

The task profile of the system allows fulfilment of all jobs by cooperation of few processes. This is important when deadlocks are to be excluded *a priori*. All system resources available for the users are pseudo-processors, which, in general, cannot occupy each other, but cooperate with equal rights in a dynamic mode through task and receipt. Nevertheless, all conditions leading to a deadlock can be satisfied [2]. However, one of them can be excluded *a priori* in this system by the 'circular wait condition'. The only prerequisite consists in checking the individual short task chains for loops.

8. Concluding remarks

The fundamental elements of CALAS70 have been working for more than one year. The system was improved steadily by extension and addition of pseudo-processors. Major difficulties have not occurred.

Experience gained under the test conditions used so far – hardware simulation of experiments – has been very satisfactory.

Further improvement now calls for measured results furnished by the real system or by simulation models, so that an even better static and, above all, dynamic adaptibility of the system can be achieved.

The small number of control commands and system states guarantee fast changes of state, good software reliability and system transparency. The given design has been described with reference to a system developed by the Institut für Datenverarbeitung in der Technik (IDT) of the Kernforschungszentrum Karlsruhe.

The concept of pseudo-processors has stood the test.

1. HERBSTREITH, H., 'A dynamic adaptive scheduling scheme for a real-time operating system', Proceedings of the 2nd European Seminar on Real-Time

Programming – Computing with Real-Time Systems, Vol. 2 (this issue), p. 84 , Erlangen (March 1972).
2. COFFMAN, E. G., et al., 'System deadlocks'. Computing Surveys, V3(2), (June 1971).
3. HERBSTREITH, H., and HEPKE, G., 'Ablaufsteuerung für ein System mit einfacher Hardware Struktur', KFK-Bericht 1530, Gesellschaft für Kernforschung Karlsruhe (1971).

Discussion

Q. When a job is waiting it can be awakened by a REPLY from a subtask or by a CALL from the Scheduler. Does it follow from this that there are two waiting states?

A. No, there is only one waiting state.

Q. Are pseudo-processors real machines?

A. No, they are virtual machines.

Q. What 'safety precautions' are there?

A. A pseudo-processor must wait until all its subtasks are finished. Each created task does its own book-keeping.

Q. Which computer is this implemented on?

A. Telefunken TR86 with 64K words of store, in which this software takes 10K.