Monograph:
Bennett, S. (1980) On Real Time System Design. Research Report. ACSE Research Report 128 . Department of Control Engineering, University of Sheffield

ON REAL-TIME SYSTEM DESIGN

by

S. BENNETT

Research Report No. 128

October 1980

Department of Control Engineering
University of Sheffield,
Mappin Street,
SHEFFIELD S1 3JD

Introduction

The past decade has produced a clarificiaiton and refinement of the techniques for the design of robust computer programs: it has also produced a greater understanding of the complex nature of the activity of 'computer programming'. The changes are reflected in the growing use of the term 'software engineering', which also reflects the growing awareness that software is a 'product' which is manufactured and sold.[1] There has been much interest in techniques for validating programs, mathematical proofs of correctness, work which has led to a clearer understanding of the differences between various types of program. Pyle,[2] drawing particularly on the work of Wirth,[3] has presented definitions identifying three types of program design:

sequential programming

multi-programming

real-time programming

These definitions are based on the kind of arguments which would be required in order to formally validate the programs.

In the classical sequential program, actions are strictly ordered as a time sequence: the behaviour of the program depends only on the effects of the individual actions and their order - the time taken to perform the actions is not of consequence. Validation therefore requires two kinds of argument:

(a) that a particular statement defines a stated action.

(b) that the various program structures produce a stated sequence of events.

A multi-task program differs from the classical sequential program in that the actions it is required to perform are not necessarily disjoint in time: several actions may be required to be performed in parallel. However, sequential relationships between the actions may still be important. The

program can be built from a number of parts (processes), which are themselves purely sequential, but which are executed concurrently and which communicate through shared variables and synchronization signals. Validation requires the arguments for sequential programs with some additions. The processes can be validated separately only if the constituent variables of each process are distinct. If variables are shared then the potential concurrency makes the effect of the program unpredictable (and hence not capable of verification) unless there is some further rule that governs the sequencing of the several actions of the processes.

The development of synchronization procedures in high level languages - ENTRY and ACCEPT in ADA- reflect the acceptance of the requirement of a sequencing rule. The need for the synchronization of processes has been knowen to program designers for a long time: the operating system for the GE CONPAC series of computers, designed in the mid 1960's, provided procedures for synchronization through its DATA TABLE access mechanisms.

It should be noted that by the use of synchronization the time taken by the individual actions are not relevant to the validity of the program. The processes can proceed at any speed, validity depends on the synchronizing procedure.

A real-time program differs from the previous types in that in addition to its actions being not necessarily disjoint in time, the sequence of some of its actions is not determined by the designer, but by the environment. They cannot be made to conform to the inter-process synchronization rules. The program can still be divided into a number of processes, but communication between the processes cannot necessarily wait for a synchronization signal- the environment process cannot be delayed. Therefore, in contrast to the previous two cases the actual time taken by an action is an essential factor in the process of validation.

Consideration of the type of reasoning necessary for validation is important not because we are seeking a method of formal proof, but because we are seeking to understand the factors which need to be considered in designing a real-time program.

It has been found by experience that the programming of real-time systems is significantly more difficult than the programming of ordinary (sequential) systems. One major difficulty has been that the major high-level languages were designed for sequential programming and hence many real-time systems were programmed using assembly level languages; the introduction of CORAL and RTL/2 has provided some improvement, but these are not truly real-time languages. A second difficulty has been a lack of a clear understanding of the significant differences between sequential, multi-process and real-time systems. These difficulties have led industrial users into one or more of the following:

(a) the development of extensive libraries of program modules written in assembly code; new installations make use existing modules,

(b) use of computer manufacturer's interpreters to implement control programs,

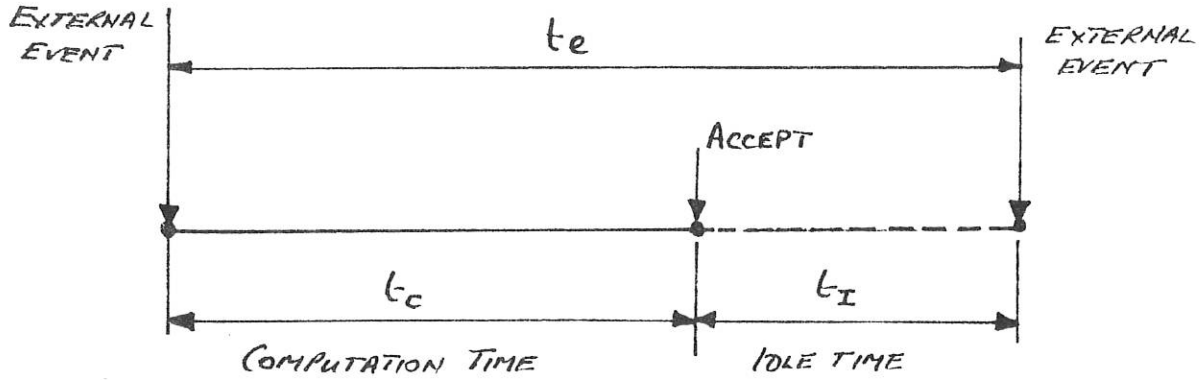
(c) design of 'in house' interpreter or compiler.

Method (a) involves a large initial investment, but provides a facility for the rapid production of reliable programs, its major disadvantages are the lack of portability and the danger of technological obsolescence - the programs are specific to a particular range of computers. In method (b) the initial investment is transferred to the system manufacturer (as are the problems of obsolescence), the disadvantages are speed of operation and lack of flexibility. Justification of method (c) is difficult: it is a luxury few can afford.[4]

Structure of real-time programs

The essential feature of real-time programs is that they are designed to run continuously and hence the natural structural element is the "infinite" closed loop. This closed loop has, however, to be synchronized to the external process. The simplest approach to the design is to structure the program as for a multi-tasking system, with synchronization by means of the ENTRY - ACCEPT mechanism, but with the added restriction that the internal processes must always be ready to ACCEPT an ENTRY, i.e. a time constraint is placed on the performance of the program actions. This is simply an alternative description of the well known polling action, and is of course the easiest type of structure to design.

The limitation of this structure is that the time taken to carry out the longest internal process must not exceed the minimum time between external events. In a simple control system with no data reduction, operator communication, or performance logging, this is also the requirement for the system to work, hence the arrangement as shown in figure 1.



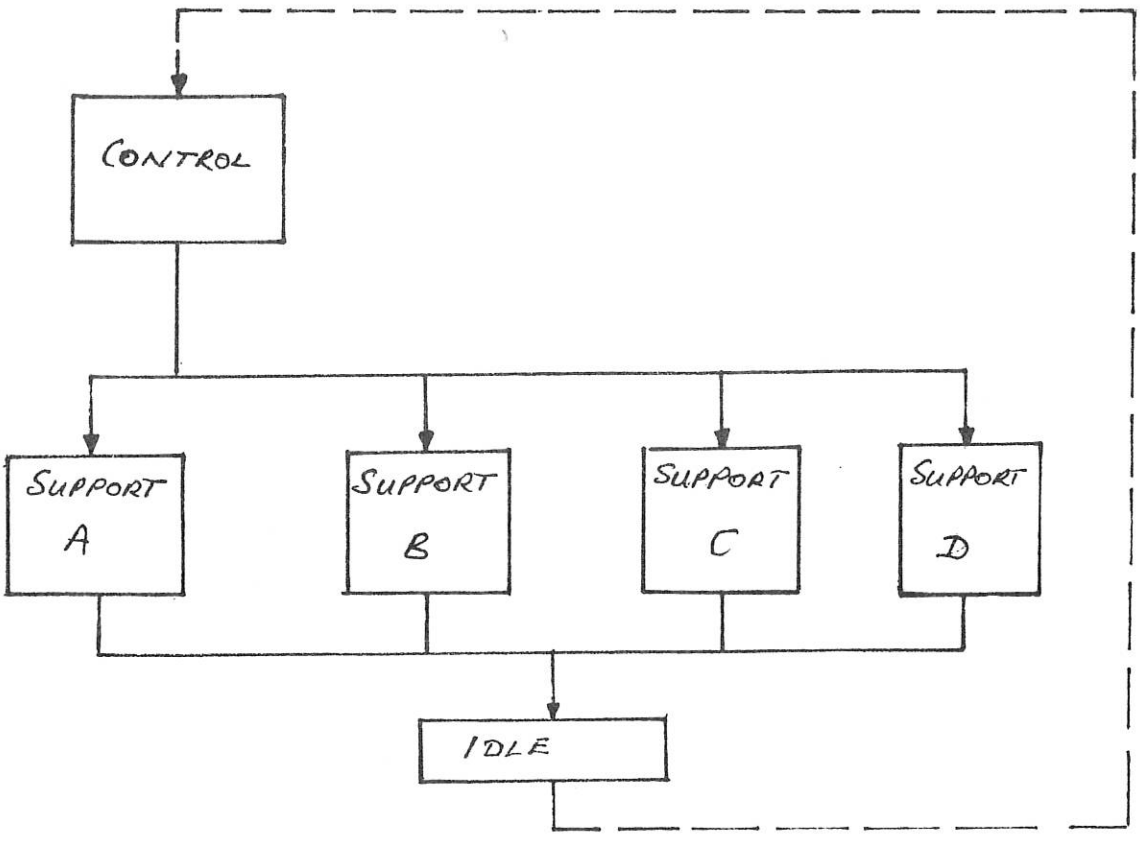
$t_e = t_c + t_I$ and $t_I > 0$ for valid program

In most real-time control systems, however, there are actions to be performed which do not have to be strictly synchronized to the external environment e.g. in a feedback control system, control action may be taken

every second, but the printing of data for the operator may only be required every 30 seconds or when requested by the operator. A real-time program, therefore, can be considered to have two types of process, the CONTROL processes and the SUPPORT processes. It is a requirement that the CONTROL processes be strictly synchronized to the EXTERNAL processes, the SUPPORT processes, however, need only by synchronized to each other and to the CONTROL process.

[The EXTERNAL process or processes may include some software: the interface between the external environment and the internal processes is a mixture of software and hardware and it is usually easier to make the boundary a software boundary. Hence the hardware interrupt will not be considered as connecting with the CONTROL process, but with an EXTERNAL process. With this approach the technical details of the implementation of the interrupts are separated from the design consideration for the CONTROL process.]

A typical arrangement is as shown in figure 2.



In terms of the synchronization procedure described above the timing diagram becomes:

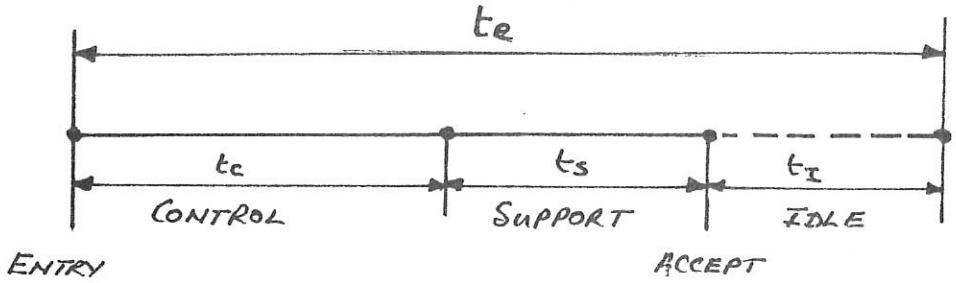


Figure 3.

amd each of the support processes has to be designed to run in less that $t_e - t_c$ seconds. This is in fact an unduly restrictive requirement, which if followed, would lead in mnay cases to unwieldly sub-division of the support tasks into many processes.

An alternative structure is shown in figure 4.

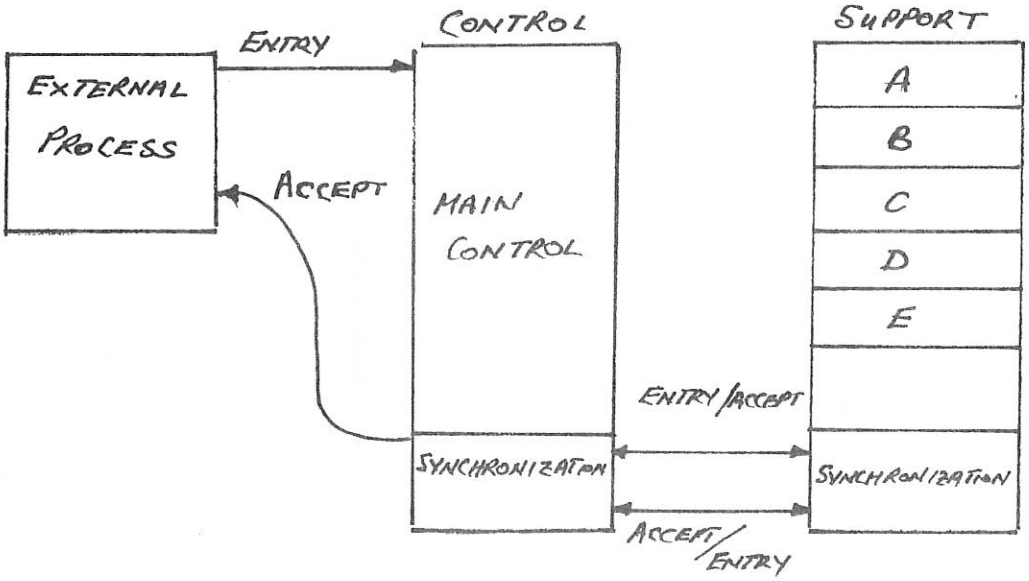


Figure 4.

In this system as soon as the main control section compilation has been completed an ACCEPT signal is given to the external process. Multi-task

operation can then proceed with the support processes. The CONTROL processes have to be designed using a real-time approach i.e. computational time is a factor to be considered, but the support processes can be designed simply as a multi-task basis. This structure does, of course, require that interrupts be used since the ACCEPT from the control program will be in the form of a permit interrupt signal and it is only through the use of interrupts that the SUPPORT processes can be freed from strict time constraints. Having organised the division in this way, the main body of the CONTROL process can be designed using sequential programming constructs, and the SUPPORT processes can be designed using multi-tasking ideas. A small segment of control section will be required to link the two.

Control-Support Communication

The separation of the program into CONTROL and SUPPORT, sections can only achieve one of its aims, the prevention of what Pyle has termed 'infection' if communication between the two sections is strictly controlled. It is possible to achieve communication by means of a shared data block as shown in figure 5.

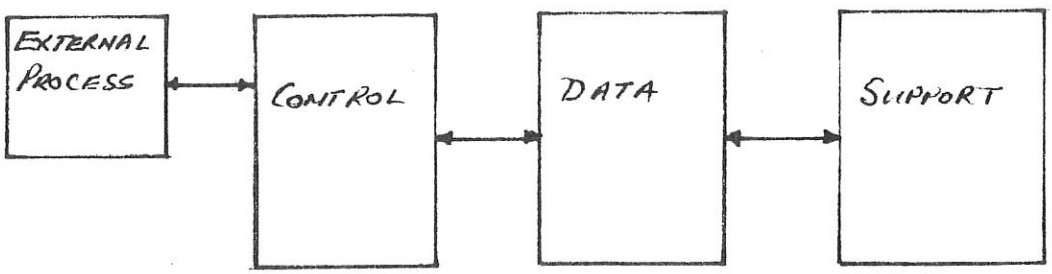


Figure 5.

and this is the way in which many systems have been designed. The behaviour of such a system is unpredictable, hence the program is not verifiable and the structure is not desirable. The unpredictable behaviour of the above system arises because the CONTROL and SUPPORT sections are not synchronized and hence either may change the data at any time without the knowledge of the

other, As an example of the danger that can that can arise from such a structure consider a section of a chemical plant in which valves are being opened and closed by a switching sequence specified by a block of data. A condition could arise in which the SUPPORT section was making a change to the sequence when an external event initiated operation of the sequence; the sequence which was performed would be partly the new sequence and partly the old and the operator would not have a clear idea of what was happening. It is of course possible, in designing the SUPPORT program, to, minimise the probability of the occurrence of such an event, but unless the SUPPORT programmer is allowed to manipulate the permit/inhibit interrupt control he cannot with certainty prevent such an event. In allowing the SUPPORT program designer control off the interrupts, his isolation from the real-time problems has been removed and the probability of infection so increased.

A much safer structure is shown in figure 6 in which two data blocks,

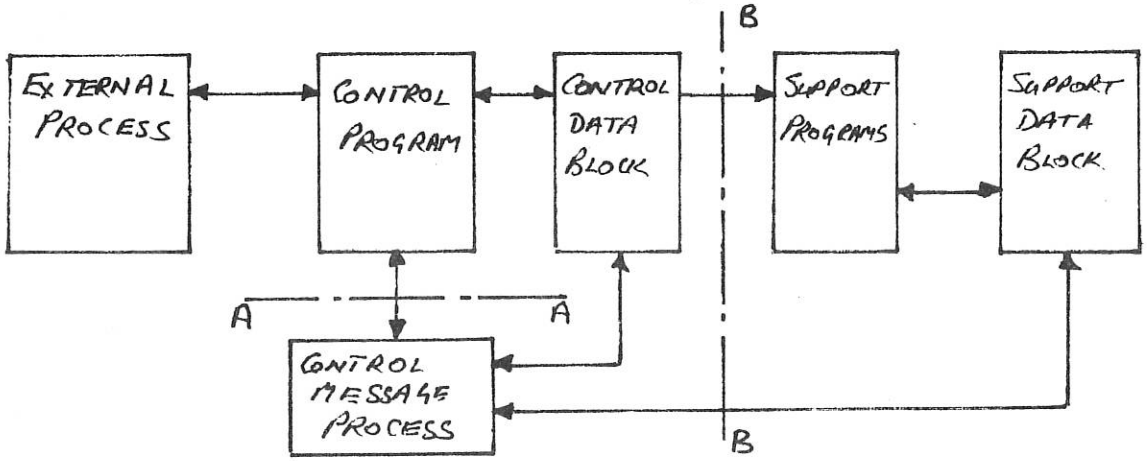


Figure 6.

CONTROL & SUPPORT are provided. The support processes are allowed to read from the CONTROL DATA BLOCK but not to write to it, all communications involving changes to the CONTROL DATA BLOCK have to pass through one process, the CONTROL MESSAGE PROCESS. In this way there is only one boundary, AA, between the essentially real-time processes and the essentially multi-task processes. The boundary BB is simply the division between two processes in a multi-program environment.

An alternative structure is shown in figure 7.

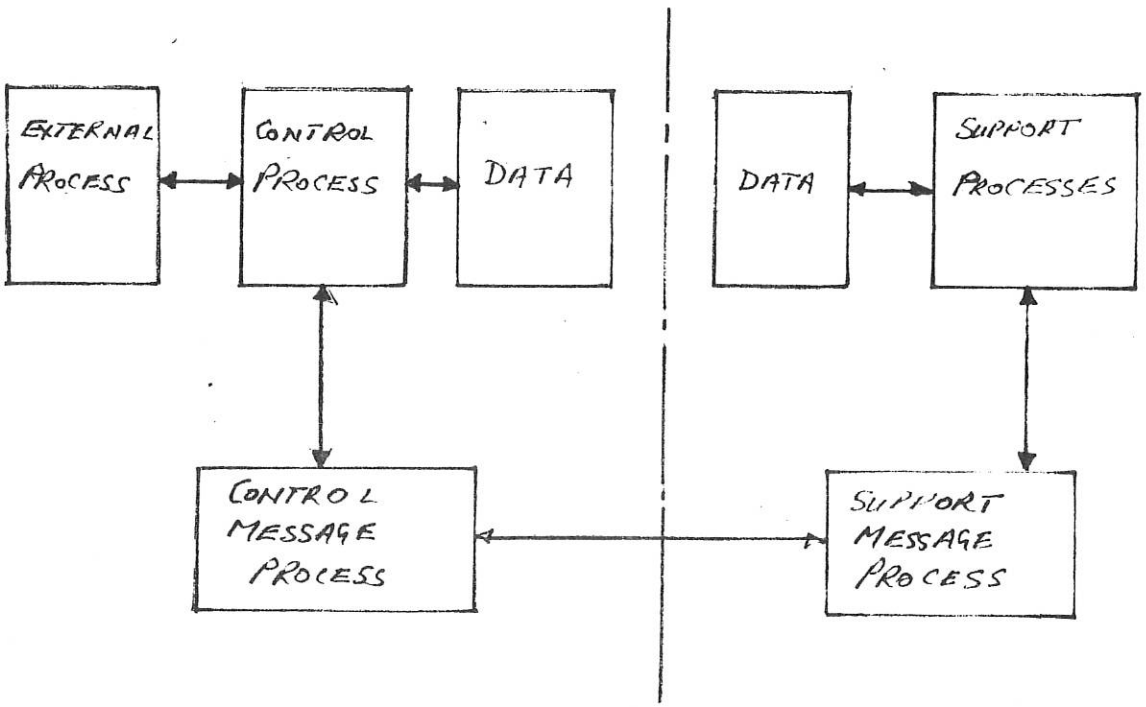


Figure 7

By structuring the system in this manner it can be seen that a natural division for distributing the real-time system over two computers is between CONTROL and SUPPORT processes. It should, however, be noted that the adoption of such a structure does not necessarily imply two computers, it can be implemented in a single computer.

Conclusion

In this note an attempt has been made to indicate the general structure required for a real-time program if full advantage is to be taken of the ideas of structured programming and synchronization of multi-programs developed over the last decade. The details of the implementation will, of course, depend on the application and the computing facilities available, but if the structure shown in figure 7 is adopted the support processes can be designed as a multi-program system; the control process can be designed as a sequantial program and the real-time processes are the EXTERNAL PROCESS HANDLER and the CONTROL MESSAGE HANDLER (better known perhaps as the interrupt handler and monitor). In this way consideration of the more difficult real-time synchronization is restricted to a small part of the design.

This structure does not, however, absolve the real-time system designer from the requirement that the real-time system run continuously. It is assumed in normal programming that program implementation is perfect - i.e. a valid program will be executed correctly. If a fault occurs the program is abandoned. A real-time system must, however, attempt to keep running, it must be fault tolerant and the designer must attempt to build into the system as much protection against corruption by faults as possible.

$$||x(t)|| \leq \frac{1}{C_2} \{\exp(C_2 t) - 1\}$$

for $t \in [0, t_1]$.

6. Conclusions

In this paper we have generalized the results of Cook (1980a,b) to the case of nonlinearly perturbed non-linear systems in both the input-output and Lyapunov approaches to the generation of state bounds. We have presented an example which shows how the two approaches bring out different aspects of the problem, although it is difficult to give a direct comparison between the methods since taking norms in different ways inevitably leads to conservative results.

7. References

1. V.M. Alekseev. An estimate for the perturbations of the solutions of ordinary differential equations (Russian) Vestnik Moskov. Univ. Ser. I. Mat. Mek. No.2. (1961), 28-36.

2. F. Brauer. Perturbations of nonlinear systems of differential equations. J. Math. Analysis App.14, (1966), 198-206.

3. P. Cook (a) On the behaviour of dynamical systems subject to bounded disturbances, Int. J. Systems Sci., (1980), 11. No.2, 159-170.

   (b) Bounds on the states of systems with bounded inputs, 3rd IMA Conference on Control Theory, University of Sheffield,(1980)

4. W. A. Coppel. Stability and Asymptotic Behaviour of Differential Equations, Heath, Boston, 1965.

5. S. M. Lozinskii. Error estimates for the numerical integration of ordinary differential equations, I. Izv. Vyss. Ucebn. Zaved. Mat. No. 5, 6(1958), 52-90 (Russian).