A dynamic adaptive scheduling scheme for a real-time operating system

H. HERBSTREITH
Kernforschungszentrum Karlsruhe, W. Germany

1. Introduction

This paper is intended to extend and complement the paper by G. Hepke [3]. It describes some design criteria and the realisation of adaptive scheduling strategies in the multiprogramming real-time system CALAS70. At an early stage in the design of the system [2] we noticed that common strategies could not be realised in the same way as in an interactive time-sharing system [1]. The reason was that the term response time in process control systems generally has another meaning from that in other systems. This forces the designer to over-design the system capacity, to avoid process failure as a result of bad observance of the required response times. On the other hand, one has to compromise for an optimal utilisation of the unused system capacities with regard to changing load distribution as encountered in laboratory automation work. The effect of the so-called background tasks on real-time requirements was eliminated by the strategies to be described below. As far as scheduling is concerned, a further requirement is the realisation of response times of the order 300-500 μsec.

2. System concept

The internal system structure is based on Dijkstra's concept of "Hierarchical Ordering of Sequential Processes" [4]. It consists of a fixed number of pseudo-processors, created during system generation. These processors execute the sequential processes. It is possible to differentiate between four classes of pseudo-processes, according to their responsibilities for the following tasks:

1. System services (e.g. I/O functions, main storage management, timer, statistics, etc.).
2. Real-time tasks.
3. Man-machine communication.
4. Non-time-critical background work.

Each pseudo-processor is identified by its ID-number. The real processor is allocated according to a second number, the priority number. The distribution of the different tasks among the pseudo-processors of classes (1), (3) and (4) follows a fixed scheme, depending on the configuration. As will be pointed out below, certain system functions require two pseudo-processors with unequal priorities. Pseudo-processors of class (2) are allocated dynamically by the scheduler. The non-time-critical background tasks are performed by only one special pseudo-processor, which executes the requesting processes on a round-robin basis with fixed time slices.

3. Scheduling strategies

For the description of the implemented scheduling concept, it seems convenient to use the distinction between two main levels of scheduling, the short-term and the medium-term scheduling, made by Saltzer [5]. Only short-term strategies are considered here.

Short-term scheduling

At this lower level of scheduling (also called hardware management) the tasks to perform are:

a. Allocation of a pseudo-processor to each sequential process.
b. Management of changes of state of the pseudo-processors.
c. Making available a set of primitive basis functions for communication between processes.

The allocation of the real processor, according to the ready list of pseudo-processors, is managed by the dispatcher. The ready list has a two-dimensional organisation and makes the selection of highest priority processes as fast as possible. One dimension, the vertical, represents the pseudo-processors. In the horizontal dimension the ready processes are queued with respect to

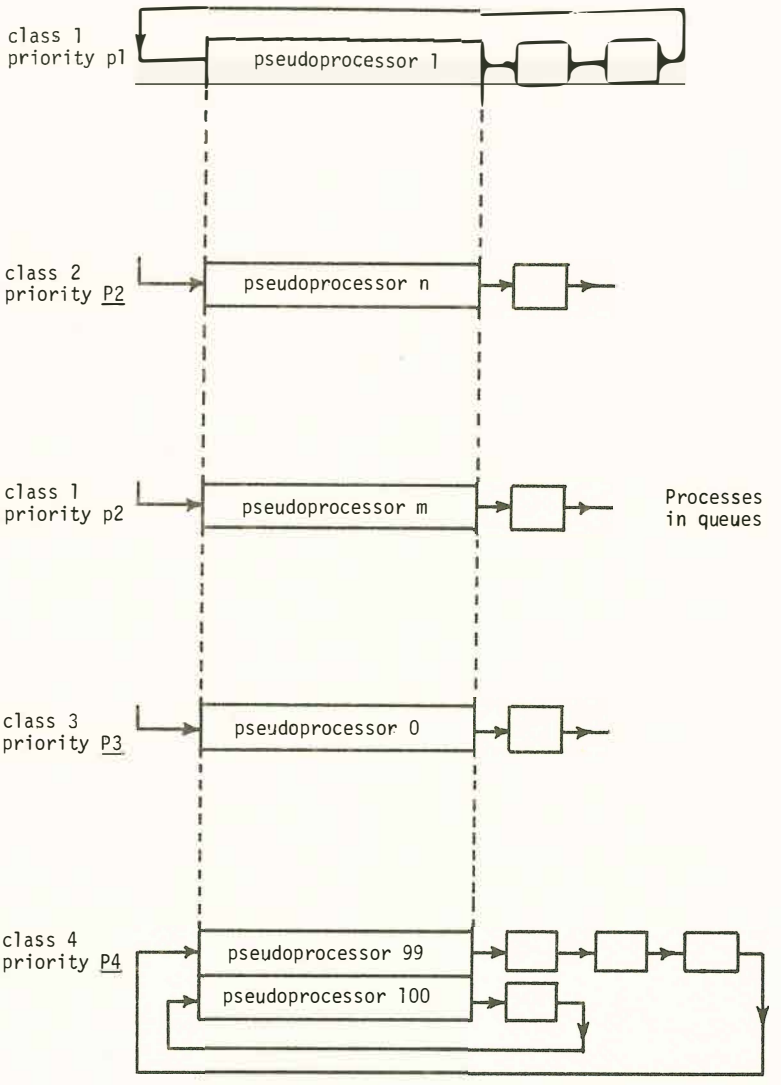

Fig. 1 Ready list of pseudo-processors

their priorities. These two degrees of freedom for scheduling allow a reasonable distribution of system capacities among the different tasks (Fig. 1).

Process queuing (Fig.2)

On creation of a process, the priority of that pseudo-processor which creates the process is assigned to it. The process will then be inserted in the corresponding queue of its pseudo-processor. The insertion is performed, according to the priority, and the look-up is started at the tail of the queue, with the restriction that a possibly already-active process at the head of this queue will not be moved by the insertion. By this method the processes with the highest priorities are always

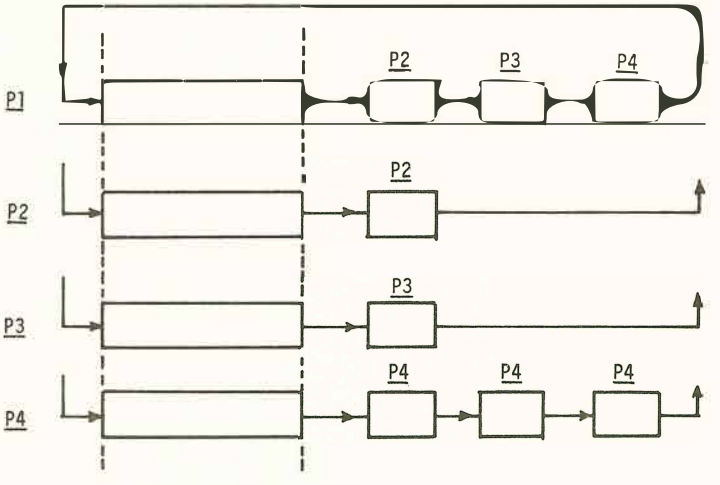

Fig. 2 Assignment of priorities to system processes

at the head of the queue and are therefore executed first. Thus, processes generated from classes (3) and (4) (Section 2) do not disturb the time sequence of processes generated from the real-time class (2).

To accomplish fast response times of the system, the scheduling functions have to be executed as fast as possible, since the real processor cannot be interrupted during the execution. As an example, we note that the necessary processor time for the insertion of a new member into a queue increases with the length of the queue. Assuming that, on an average, ten members will have to be checked, this time should not exceed 300-500 μsec. This assumption is sufficient, and so the length of all queues connected to class (1) will be reduced in the following way:

a. If the length of the queue exceeds 15, all processes generated by classes (3) and (4) are rejected.
b. If the length still exceeds a value of 20, the lowest priorities of the real-time class are rejected too.

Assignment of pseudo-processors to system service processes

As already mentioned, there are two pseudo-processors available for each of some special dedicated system service functions of class (1). The corresponding processes in the queues are assigned dynamically to either one or the other pseudo-processors, depending on the priority of these processes. If $\underline{P}_k$ (with $k = 2, 3, 4$) are the priorities of classes k respectively, and p_1, p_2 the priorities of the special pseudo-processors of class (1), then (Fig. 3)

$$p_1 > \underline{P}_2 > p_2 > \underline{P}_3 > \underline{P}_4$$

The pseudo-processor with priority p_2 is always executing processes as long as there are no processes generated from class (2). On creation of processes of class (2), the pseudo-processor with priority p_1, will be switched on until all these real-time tasks have been executed.

Three essential actions are required by the processor switching:

a. The last active processor is removed from ready list and becomes invisible for the dispatcher.
b. The status of the now blocked processor is copied into the register field of the other processor.
c. The new processor to activate is entered into the ready list and the corresponding process queue is assigned to it.

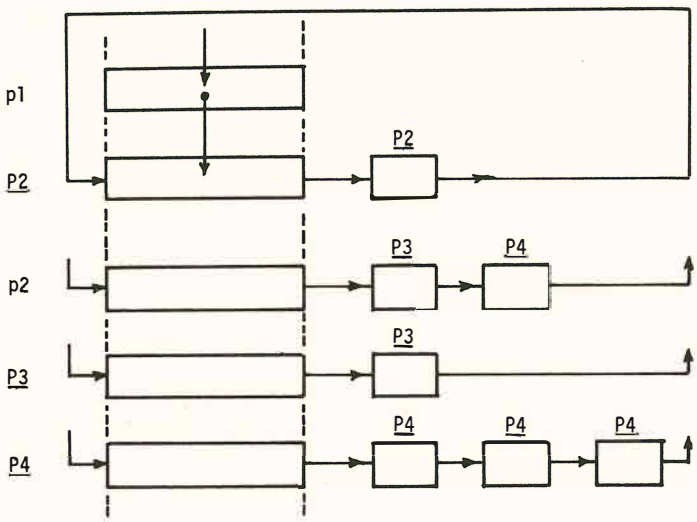

Fig. 3 Allocation of pseudo-processors p2 to system processes

4. Conclusion

By the use of these three scheduling strategies we have been able to develop standard solutions for the design of a real-time scheduler. We have shown the possibility of isolating real-time work from background tasks with a parallel decrease in response time. It is our belief that the methods discussed here improve the design of central multi-access systems in process control and laboratory automation. Our concept allows for the introduction of additional background work into the system, without deterioration of real-time requirements.

1. CHAMBERLIN, D., 'Comparative modelling of time-slicing and deadline scheduling for interactive systems', RC 3378 (15410), Research Report of the IBM Thomas Watson Research Center, Yorktown Heights, New York 10598 (25 May 1971).
2. GAGEL, G., HEPKE, G., HERBSTREITH, H., and NEHMER, J., 'CALAS68 – ein computergestütztes Vielfachzugriffssystem zur Laborautomatisierung', Ext. Bericht Nr. 19/69-1 der Gesellschaft für Kernforschung Karlsruhe (November 1970).
3. HEPKE, G., 'CALAS70 – a real-time operating system based on pseudo-processors', Proceedings of the 2nd European Seminar on Real-Time Programming – Computing with Real-Time Systems, Vol. 2 (this issue), p. 81 , Erlangen (March 1972).
4. DIJKSTRA, E.W., 'Hierarchical ordering of sequential processes', Intern. Seminar on Operating System Techn., Belfast (August 1971).
5. SALTZER, J.H., 'Traffic control in a multiplexed compiler system', MAC-TR-30, MIT, Cambridge (July 1966).