Workshop on components of real-time systems

Chairman: **Prof. Dr R. BAUMAN**
Munich, W. Germany

Discussion was concerned with the use of priorities for controlling the task scheduling and working space allocation (how dynamic it has to be). It was generally agreed that a real-time system differed from a general time-sharing system in the attitude of the users:

J. Ours is not a time-sharing system in which users may be trying to break the system. We can assume they are cooperating on the solution of a problem.

Priority structures

L. There is a golden rule: the priority of a process varies inversely as the critical response time for that process. Demands for very short response time may call for activation of lower priority processes. As an example, a block of disk transfers generates demands on at least 3 levels of priority:

1. The demand for individual word transfer.
2. The interrupt to identify the end of the block.
3. The scheduling of the process which will use the information.

The level of priority decreases from 1 to 3.

G. Hardware interrupts are handled at the priority needed to deal with the necessary immediate action only. A software interrupt is then scheduled at a lower priority to deal with the less urgent part of the action. In general the higher priorities are used for the minimum of time to handle the most urgent part of a response, the rest of the processing being scheduled at a lower priority.

J. Is it not true that the author of a task is the best one to assign its priority, since he knows how important his job is. (This is not a time-sharing system for user-programmers.) We insist that programs suspend themselves regularly stating their own priority, e.g. low, high, resume after 5 secs.

Working space allocation

S. In relation to Dr Pyle's paper I understand that what he is proposing with regard to store allocation is as follows: store is allocated in a static manner at load time for each process but can be allocated dynamically within a process.

Q. What is 'load time' in a real-time system?

A. When you first set the program up to run.

M. The choice of storage allocation mechanism depends very much on the type of application. The term 'real-time' is used to cover a multitude of sins from a simple data gathering system which has fixed storage requirements but relatively fast response times, to an Airline reservation system which needs sophisticated algorithms for storage but not particularly demanding response times.

Y. Cases can be made for dynamic storage allocation, for example a free store area containing a pool of blocks of fixed size (the size agreed between the designers of the processes using this pool). However, even with this in a system, the whole store is partitioned in a fixed way at load time between those areas used for program storage, those used for genuine static storage and those used for stacks or buffer pools. The management of the areas within which storage is used dynamically is done by a higher level program at a level of sophistication above that of the primitives given.

P. Dynamic allocation of working space (administered by the operating system) can give efficient use of store and easier implementation of re-entrant tasks. At RRE we implemented message passing in an operating system. The mechanism provides message communication between parallel processes, which simplifies the problems of controlling program interactions. We found that the timing overheads in acquiring free areas and sending them to other processes were rather high.

We are now in the process of implementing a hardware version of this system. The hardware, which will be a computer peripheral, will enable us to acquire free area and send messages to other processes in time of order 100 nsecs.

Y. We may contrast the situation when the time a piece of program will take to execute can be determined, perhaps as a conditional expression involving conditions not removable in advance (e.g. a program containing a test on character read in and appropriate action, or a program containing an 'attempt to secure' primitive), with the situation when that time cannot be determined even to that extent, and is essentially unbounded (e.g. a program containing a 'secure' primitive, or a program in an environment using dynamic storage allocation where there is the possibility that store required may not be immediately available).

I regard the former situation, of conditionally determinable and bounded time, as acceptable for real-time systems, and the latter, of unpredictable unbounded time, as not acceptable.