Computer Science 773

Robotics and Real-time Control

RATE-MONOTONIC ANALYSIS

In designing the computer systems for real-time control, *scheduling* is an exceedingly important task. Typically, the controller runs several (perhaps very many) processes, most of which must be executed within more or less precisely defined time limits, with the definition the more precise for harder real-time operations.

Scheduling itself is not a new problem, and operating systems have been scheduling processes for many years with varying degrees of sophistication. The most impressive scheduling performance was achieved with rather old-fashioned batch processing systems, where very high processing efficiencies were attained. When the systems were developed, high efficiency was very important because processing hardware was very expensive. The batch systems work because they require a lot of information about the processes before accepting them : an estimate of the processor time required, whether or not other devices will be used, and how much, and so on. Given this sort of information, it is possible to choose a set of processes to run together which will make better use of the available facilities than any of the processes individually. Without the information, as with more recent systems in which timesharing or other unpredictable processing modes are common, far lower levels of efficiency are achieved, but that doesn't matter a lot because processors are now comparatively very cheap.

With real-time systems, the requirement for careful scheduling returns, but the reason is no longer the expense of the processor; instead, it is the real-time constraints on the process, which must be satisfied if the process is to be controlled successfully. The methods used in operating systems are not in themselves sufficient to do the job; operating systems are intended to get work done as quickly as possible, but not to meet deadlines. The constraints they satisfy are to do with preserving the order of operations, but not with explicit timing.

One solution is to provide a processor for each process. Even with cheap processor, this can turn out to be uncomfortably expensive for a large system, and the additional communications facilities needed can be a nuisance, increasing bulk, weight, timing delays, and cost. Additional hardware and software is required, with consequent increases in the probability of failure, and for tightly linked processes (such as descendants from the same root process) the separation might well be more difficult to implement than processor sharing. All in all, while it is not uncommon to use several processors, going to extremes seems to be unproductive in various ways, and the problem of scheduling several processes within a processor remains.

As with the operating systems, the solution lies in getting information about the processes; and, as the constraints are much more severe than those on operating systems, one might reasonably expect that more detailed information would be required. One would not be disappointed. Several possible approaches to real-time scheduling have been explored from time to time, but the winner so far appears to be *rate-monotonic scheduling.* This method was formulated in the early 1990s, and combines a well defined analysis procedure with a (limited, but not very limited) guarantee of performance.

CLASSIFYING THE TASKS.

The first job is to identify the tasks (processes) which are to be performed, and to gather information on their requirements.

Three types of task are defined, a task's classification depending on its timing requirements. All are typical of realistic processing requirements of real-time systems. The task types, in order of decreasingly disciplined behaviour, are :

- **Periodic tasks**, which must be scheduled at regular intervals, and for which precise estimates of computational requirements can be given. These tasks are commonly concerned with sampling data and performing computations on the sampled data.

- **Sporadic tasks**, which occur at irregular intervals, but are bounded in size so have hard deadlines. They can be guaranteed to occur not more frequently than some fixed limit. These tasks are typically responses to external events such as interrupts. As a specific interrupt is normally tied to some specific external event (once every rotation of a wheel, for example), the physical properties of the plant impose limits on the maximum frequency.

- **Aperiodic tasks**, also irregular, with no constraints on the time between successive task initiations, but still bounded in time. Timing is determined by probabilities rather than mechanical constraints, and a finite average rate of arrival is required. Examples are processes which deal with interrupts from independent distant sources which are not correlated with the local machinery. These are the most awkward tasks which can be handled by the method.

By restricting the method to this entirely reasonable set of task types, the really disruptive events – many tasks arriving almost simultaneously with very large processing requirements – are eliminated.

SCHEDULING.

Most information is required about the periodic tasks, because these form the background against which tasks of the other types must be scheduled. Four quantities must be defined for task number i :

C_i	Computation time	The processing time required at each invocation.
T_i	Period	The time between successive invocations.
D_i	Deadline	The time within the period by which the task must be complete.
I_i	Phase	The time within the period before which the task must not be started.

These values must conform to a number of obvious constraints, which can be checked. The approach taken is then to draw up a schedule for the periodic tasks, within which there is enough slack time in the right places to ensure that the sporadic tasks can be scheduled, and that there is a good chance (which can be made as good as you like) that the aperiodic tasks can also be scheduled.

Alan Creak,
March, 1997.