Computer Science 773

Robotics and Real-time Control

CONTROL SYSTEM ARCHITECTURES

Very early in the development of computer control systems, it became clear that some systematic overall organisation was necessary. The obvious approach of writing something like a device driver to look after each of your machines and then leaving them alone to do so, just as you might for the peripheral devices in a conventional operating system, simply didn't work; some devices would get far too much attention, while others would get too little.

The difference is that in an ordinary system there is some sort of programme running which determines when the devices must operate, and generally coordinates their activities. Clearly, we need some sort of supervisor to manage the machines controlled by a computer used for machine control. This must make sure that each machine under its care gets attention when it needs attention, and sufficiently frequently – recall the sampling theorem – to keep the system up to date and running smoothly.

However it's done, it must cater for any sort of attached device – which, in effect, means that it must be able to handle any sort of signal from the device and send any sort of signal to the device. Output to the devices is rarely much trouble, but input can be difficult to handle. Some devices produce signals of events which persist, some produce signals which last only for a moment, then vanish, so must be handled as interrupts, some produce nothing at all, and have to be polled regularly to find out what they're doing.

THE CYCLIC EXECUTIVE.

The first organised and reasonably standardised method for dealing with the problem was the cyclic executive. In this system, the computer gives attention to each device – or, more precisely, to each input or output connection – in turn at regular intervals, so the basic system structure is a simple loop which is controlled by a clock so that it starts at regular intervals.

Interrupts are obviously not handled simply by such a structure. The interrupt itself must still be dealt with when it arrives, but it must be handled quickly by a small interrupt service routine which records the occurrence of the interrupt and performs any immediately required reading or writing. The software driving the device must then check the interrupt flag when its turn to execute comes round.

Another requirement not well managed by the simple scheme is that of differing frequencies of polling. It is not unusual to have functions within a system which require very different response speeds; an interrupt might require attention within a few milliseconds, while a thermostat need only be checked every minute or so. This disparity can be handled by running several nested cycles, with the fastest on the "outside", and successive inner cycles executed once every (say) 100 times round the outer cycle. A difficulty with this simple-minded technique is that if there are many tasks to be accomplished on the longer time scale then there is far more than usual to do in the hundredth execution of the outer loop, and it might be impossible to meet the loop deadline. There are obvious answers to that objection, but it illustrates that care must be taken to meet the real-time constraints, and perhaps that solutions which seem obvious in ordinary programming terms don't necessarily work in real time. (Recall Wirth's hope that you could "bolt on" a real-time module to concurrent programming to make it all work.)

Nevertheless, provided that it satisfies the timing constraints, a real-time executive is a very satisfactory system architecture. It is simple and reliable; it is clear that all the items in the loops are being dealt with at well defined frequencies. On the other hand, it can be inefficient, as every item is checked each time through its loop whether or not there is work to be done; and it can be difficult to guarantee satisfactory response to interrupts, particularly if they require a significant amount of computing and a quick response. The real-time executive is also not very good at dealing with systems in which the different parts require different amounts of attention at different times; it's best for a plant in which each item in the loop requires a little work each time round.

INTERRUPT-DRIVEN SYSTEMS.

At the other end of the scale, there are interrupt-driven systems. These are good at catering for quick responses and irregular work patterns, but it is much harder to exercise overall control. Because of that, more careful analysis of the system controlled is needed before designing an interrupt-driven controller, as it must be certain that it will be possible to give the necessary level of support to the various activities.

REALISTIC SYSTEMS.

A wholly interrupt-driven system is not usually effective unless the overall processing load is very light, or the interrupts can somehow be guaranteed not to interfere with each other. In addition, most practical cases do require some proportion of regular polling to monitor plant conditions or other slowly changing factors, and these do not have naturally associated interrupts. Practical systems are therefore likely to contain some mixture of interrupt-driven and polled activities, but in such a case it is difficult to demonstrate that the system cannot be overloaded, as it is always possible for a sudden rush of interrupts to delay the polling cycle to an unacceptable extent.

Over the past decade or so, there has been significant progress in the design of such mixed systems. A general theme has been to use the extensive knowledge about the system, and in particular about the known set of processes which will be running, to classify the various activities, and to take into account the sorts of behaviour they exhibit when devising an operating schedule.

Alan Creak,
April, 1998.