Real-Time Systems

From OSDev Wiki

Real-Time Operating Systems are systems in which certain processes or operations have guaranteed minimum and/or maximum response times. That is to say, the system ensures that it will complete operation x after time **t** but before time $\mathbf{t_2}$, whatever **t** and $\mathbf{t_2}$ are, without fail, even at the expense of other lower priority operations.

Speed, in and of itself, is not critical; the primary goal is predictability. A response time less than *t* may be just as bad as, or worse than, one greater than *t'*.

'Soft' real-time systems (e.g., laser-printer microcontrollers) promise best-effort reliability; 'hard' real-time systems (e.g., 'fly-by-wire' avionics in an aircraft) need to have zero-failure-tolerance reliability.

Real time operating systems are, as a general rule, only used in time-dependent embedded applications; general-purpose systems rarely if ever need to meet real-time constraints. When the do occur, real-time considerations may rule out the use of some common techniques, such as virtual memory, which may make the system's behavior less deterministic. Best-effort real-time functionality can however be useful in general purposes systems for supporting the needs of applications like digital audio workstations which demand both reliability (live recording) and low latency (real-time synthesis/processing), often at the same time.

One of the best-known Real Time OS for the x86 platform is QnX. Each system call of QnX is documented with a 'worst case completion time'.

It should be noted that "being Real Time" not necessarily means that an OS is very good at playing MPEGs, or using hardware efficiently - this is a common misunderstanding. To the contrary, for a system to provide hard real-time services implies that it can use only a limited percentage of the system's resources, including CPU time. It also fundamentally changes how software can be built for the system. For example, Rate Monotonic Scheduling - a hard real-time scheduling algorithm - can guarantee time restraints only up to 70% CPU load. Beyond that, the system has "to hit the red button" because it can no longer guarantee anything.

This implies that applications have to state their run-time requirements beforehand - how often they must be called in a second, which maximum response time is acceptable etc. All this information must be provided by the application programmer. In some cases the information is provided in implicit form, for example by arranging processes to have a certain order of priorities which allows them to meet their goals.

Bottom line, hard real time is for industrial, medical, or military systems. On your average desktop, it is misplaced.

Related Articles

- Real-Time Scheduling Algorithms
- Wikipedia - Real Time Operating System
- Papers describing the real time implementation in Plan 9 from Bell Labs (http://doc.cat-v.org/plan_9/real_time/)