WIKIPEDIA

Real-time computing

Real-time computing (**RTC**), or **reactive computing** is the computer science term for hardware and software systems subject to a "real-time constraint", for example from event to system response.[1] Real-time programs must guarantee response within specified time constraints, often referred to as "deadlines".[2]

Real-time responses are often understood to be in the order of milliseconds, and sometimes microseconds. A system not specified as operating in real time cannot usually *guarantee* a response within any timeframe, although *typical* or *expected* response times may be given. Real-time processing *fails* if not completed within a specified deadline relative to an event; deadlines must always be met, regardless of system load.

A real-time system has been described as one which "controls an environment by receiving data, processing them, and returning the results sufficiently quickly to affect the environment at that time".[3] The term "real-time" is also used in simulation to mean that the simulation's clock runs at the same speed as a real clock, and in process control and enterprise systems to mean "without significant delay".

Real-time software may use one or more of the following: synchronous programming languages, real-time operating systems, and real-time networks, each of which provide essential frameworks on which to build a real-time software application.

Systems used for many mission critical applications must be real-time, such as for control of fly-by-wire aircraft, or anti-lock brakes, both of which demand immediate and accurate mechanical response.[4]

Contents

- History
- Criteria for real-time computing
  - Real-time in digital signal processing
    - Live vs. real-time
- Real-time and high-performance
- Near real-time
- Design methods
- See also
- References
- Further reading
- External links

History

The term *real-time* derives from its use in early simulation, in which a real-world process is simulated at a rate that matched that of the real process (now called real-time simulation to avoid ambiguity). Analog computers, most often, were capable of simulating at a much faster pace than

real-time, a situation that could be just as dangerous as a slow simulation if it were not also recognized and accounted for.

Minicomputers, particularly in the 1970s onwards, when built into dedicated embedded systems such as DOG (Digital on-screen graphic) scanners, increased the need for low-latency priority-driven responses to important interactions with incoming data and so operating systems such as Data General's RDOS (Real-Time Disk Operating System) and RTOS with background and foreground scheduling as well as Digital Equipment Corporation's RT-11 date from this era. Background-foreground scheduling allowed low priority tasks CPU time when no foreground task needed to execute, and gave absolute priority within the foreground to threads/tasks with the highest priority. Real-time operating systems would also be used for time-sharing multiuser duties. For example, Data General Business Basic could run in the foreground or background of RDOS and would introduce additional elements to the scheduling algorithm to make it more appropriate for people interacting via dumb terminals.

Once when the MOS Technology 6502 (used in the Commodore 64 and Apple II), and later when the Motorola 68000 (used in the Macintosh, Atari ST, and Commodore Amiga) were popular, anybody could use their home computer as a real-time system. The possibility to deactivate other interrupts allowed for hard-coded loops with defined timing, and the low interrupt latency allowed the implementation of a real-time operating system, giving the user interface and the disk drives lower priority than the real-time thread. Compared to these the programmable interrupt controller of the Intel CPUs (8086..80586) generates a very large latency and the Windows operating system is neither a real-time operating system nor does it allow a program to take over the CPU completely and use its own scheduler, without using native machine language and thus surpassing all interrupting Windows code. However, several coding libraries exist which offer real time capabilities in a high level language on a variety of operating systems, for example Java Real Time. The Motorola 68000 and subsequent family members (68010, 68020 etc.) also became popular with manufacturers of industrial control systems. This application area is one in which real-time control offers genuine advantages in terms of process performance and safety.

Criteria for real-time computing

A system is said to be *real-time* if the total correctness of an operation depends not only upon its logical correctness, but also upon the time in which it is performed.[5] Real-time systems, as well as their deadlines, are classified by the consequence of missing a deadline:[6]

- *Hard* – missing a deadline is a total system failure.
- *Firm* – infrequent deadline misses are tolerable, but may degrade the system's quality of service. The usefulness of a result is zero after its deadline.
- *Soft* – the usefulness of a result degrades after its deadline, thereby degrading the system's quality of service.

Thus, the goal of a *hard real-time system* is to ensure that all deadlines are met, but for *soft real-time systems* the goal becomes meeting a certain subset of deadlines in order to optimize some application-specific criteria. The particular criteria optimized depend on the application, but some typical examples include maximizing the number of deadlines met, minimizing the lateness of tasks and maximizing the number of high priority tasks meeting their deadlines.

Hard real-time systems are used when it is imperative that an event be reacted to within a strict deadline. Such strong guarantees are required of systems for which not reacting in a certain interval of time would cause great loss in some manner, especially damaging the surroundings physically or threatening human lives (although the strict definition is simply that missing the deadline constitutes failure of the system). Some examples of hard real-time systems:

- A car engine control system is a hard real-time system because a delayed signal may cause engine failure or damage.
- Medical systems such as heart pacemakers. Even though a pacemaker's task is simple, because of the potential risk to human life, medical systems like these are typically required to undergo thorough testing and certification, which in turn requires hard real-time computing in order to offer provable guarantees that a failure is unlikely or impossible.
- Industrial process controllers, such as a machine on an assembly line. If the machine is delayed, the item on the assembly line could pass beyond the reach of the machine (leaving the product untouched), or the machine or the product could be damaged by activating the robot at the wrong time. If the failure is detected, both cases would lead to the assembly line stopping, which slows production. If the failure is not detected, a product with a defect could make it through production, or could cause damage in later steps of production.
- Hard real-time systems are typically found interacting at a low level with physical hardware, in embedded systems. Early video game systems such as the Atari 2600 and Cinematronics vector graphics had hard real-time requirements because of the nature of the graphics and timing hardware.
- Softmodems replace a hardware modem with software running on a computer's CPU. The software must run every few milliseconds to generate the next audio data to be output. If that data is late, the receiving modem will lose synchronization, causing a long interruption as synchronization is reestablished or causing the connection to be lost entirely.
- Many types of printers have hard real-time requirements, such as inkjets (the ink must be deposited at the correct time as the printhead crosses the page), laser printers (the laser must be activated at the right time as the beam scans across the rotating drum), and dot matrix and various types of line printers (the impact mechanism must be activated at the right time as the print mechanism comes into alignment with the desired output). A failure in any of these would cause either missing output or misaligned output.

In the context of multitasking systems the scheduling policy is normally priority driven (pre-emptive schedulers). In some situations, these can guarantee hard real-time performance (for instance if the set of tasks and their priorities is known in advance). There are other hard real-time schedulers such as rate-monotonic which is not common in general-purpose systems, as it requires additional information in order to schedule a task: namely a bound or worst-case estimate for how long the task must execute. Specific algorithms for scheduling such hard real-time tasks exist, such as earliest deadline first, which, ignoring the overhead of context switching, is sufficient for system loads of less than 100%.[7] New overlay scheduling systems, such as an adaptive partition scheduler assist in managing large systems with a mixture of hard real-time and non real-time applications.

Firm real-time systems are more nebulously defined, and some classifications do not include them, distinguishing only hard and soft real-time systems. Some examples of firm real-time systems:

- The assembly line machine described earlier as *hard* real-time could instead be considered *firm* real-time. A missed deadline still causes an error which needs to be dealt with: there might be machinery to mark a part as bad or eject it from the assembly line, or the assembly line could be stopped so an operator can correct the problem. However, as long as these errors are infrequent, they may be tolerated.

Soft real-time systems are typically used to solve issues of concurrent access and the need to keep a number of connected systems up-to-date through changing situations. Some examples of soft real-time systems:

- Software that maintains and updates the flight plans for commercial airliners. The flight plans must be kept reasonably current, but they can operate with the latency of a few seconds.
- Live audio-video systems are also usually soft real-time. A frame of audio that's played late may cause a brief audio glitch (and may cause all subsequent audio to be delayed correspondingly, causing a perception that the audio is being played slower than normal), but this may be better

than the alternatives of continuing to play silence, static, a previous audio frame, or estimated data. A frame of video that's delayed typically causes even less disruption for viewers. The system can continue to operate and also recover in the future using workload prediction and reconfiguration methodologies.[8]

- Similarly, video games are often soft real-time, particularly as they try to meet a target frame rate. As the next image cannot be computed in advance, since it depends on inputs from the player, only a short time is available to perform all the computing needed to generate a frame of video before that frame must be displayed. If the deadline is missed, the game can continue at a lower frame rate; depending on the game, this may only affect its graphics (while the gameplay continues at normal speed), or the gameplay itself may be slowed down (which was common on older third- and fourth-generation consoles).

Real-time in digital signal processing

In a real-time digital signal processing (DSP) process, the analyzed (input) and generated (output) samples can be processed (or generated) continuously in the time it takes to input and output the same set of samples *independent* of the processing delay.[9] It means that the processing delay must be bounded even if the processing continues for an unlimited time. That means that the mean processing time per sample, including overhead, is no greater than the sampling period, which is the reciprocal of the sampling rate. This is the criterion whether the samples are grouped together in large segments and processed as blocks or are processed individually and whether there are long, short, or non-existent input and output buffers.

Consider an audio DSP example; if a process requires 2.01 seconds to analyze, synthesize, or process 2.00 seconds of sound, it is not real-time. However, if it takes 1.99 seconds, it is or can be made into a real-time DSP process.

A common life analogy is standing in a line or queue waiting for the checkout in a grocery store. If the line asymptotically grows longer and longer without bound, the checkout process is not real-time. If the length of the line is bounded, customers are being "processed" and output as rapidly, on average, as they are being inputted then that process *is* real-time. The grocer might go out of business or must at least lose business if they cannot make their checkout process real-time; thus, it is fundamentally important that this process is real-time.

A signal processing algorithm that cannot keep up with the flow of input data with output falling farther and farther behind the input, is not real-time. But if the delay of the output (relative to the input) is bounded regarding a process that operates over an unlimited time, then that signal processing algorithm is real-time, even if the throughput delay may be very long.

Live vs. real-time

Real-time signal processing is necessary, but not sufficient in and of itself, for live signal processing such as what is required in live event support. Live audio digital signal processing requires both real-time operation and a sufficient limit to throughput delay so as to be tolerable to performers using stage monitors or in-ear monitors and not noticeable as lip sync error by the audience also directly watching the performers. Tolerable limits to latency for live, real-time processing is a subject of investigation and debate but is estimated to be between 6 and 20 milliseconds.[10]

Real-time bidirectional telecommunications delays of less than 300 ms ("round trip" or twice the unidirectional delay) are considered "acceptable" to avoid undesired "talk-over" in conversation.

Real-time and high-performance

Real-time computing is sometimes misunderstood to be high-performance computing, but this is not an accurate classification.[11] For example, a massive supercomputer executing a scientific simulation may offer impressive performance, yet it is not executing a real-time computation. Conversely, once the hardware and software for an anti-lock braking system have been designed to meet its required deadlines, no further performance gains are obligatory or even useful. Furthermore, if a network server is highly loaded with network traffic, its response time may be slower but will (in most cases) still succeed before it times out (hits its deadline). Hence, such a network server would not be considered a real-time system: temporal failures (delays, time-outs, etc.) are typically small and compartmentalized (limited in effect) but are not catastrophic failures. In a real-time system, such as the FTSE 100 Index, a slow-down beyond limits would often be considered catastrophic in its application context. The most important requirement of a real-time system is consistent output, not high throughput.

Some kinds of software, such as many chess-playing programs, can fall into either category. For instance, a chess program designed to play in a tournament with a clock will need to decide on a move before a certain deadline or lose the game, and is therefore a real-time computation, but a chess program that is allowed to run indefinitely before moving is not. In both of these cases, however, high performance is desirable: the more work a tournament chess program can do in the allotted time, the better its moves will be, and the faster an unconstrained chess program runs, the sooner it will be able to move. This example also illustrates the essential difference between real-time computations and other computations: if the tournament chess program does not make a decision about its next move in its allotted time it loses the game—i.e., it fails as a real-time computation—while in the other scenario, meeting the deadline is assumed not to be necessary. High-performance is indicative of the amount of processing that is performed in a given amount of time, whereas real-time is the ability to get done with the processing to yield a useful output in the available time.

Near real-time

The term "near real-time" or "nearly real-time" (NRT), in telecommunications and computing, refers to the time delay introduced, by automated data processing or network transmission, between the occurrence of an event and the use of the processed data, such as for display or feedback and control purposes. For example, a near-real-time display depicts an event or situation as it existed at the current time minus the processing time, as nearly the time of the live event.[12]

The distinction between the terms "near real time" and "real time" is somewhat nebulous and must be defined for the situation at hand. The term implies that there are no significant delays.[12] In many cases, processing described as "real-time" would be more accurately described as "near real-time".

Near real-time also refers to delayed real-time transmission of voice and video. It allows playing video images, in approximately real-time, without having to wait for an entire large video file to download. Incompatible databases can export/import to common flat files that the other database can import/export on a scheduled basis so that they can sync/share common data in "near real-time" with each other.

The distinction between "near real-time" and "real-time" varies, and the delay is dependent on the type and speed of the transmission. The delay in near real-time is typically in a range of 1-10 seconds.

Design methods

Several methods exist to aid the design of real-time systems, an example of which is MASCOT, an old but very successful method which represents the concurrent structure of the system. Other examples are HOOD, Real-Time UML, AADL, the Ravenscar profile, and Real-Time Java.

See also

- Autonomous peripheral operation
- DSOS
- Processing modes
- Ptolemy Project
- Real-time data
- Real-time computer graphics
- Real-time testing
- Scheduling analysis real-time systems
- Synchronous programming language
- Time-utility function
- Worst-case execution time

References

1. "FreeRTOS - Open Source RTOS Kernel for small embedded systems - What is FreeRTOS FAQ?" (https://www.freertos.org/FAQWhat.html#WhyUseRTOS). *FreeRTOS*. Retrieved 2021-03-08.
2. Ben-Ari, Mordechai; "Principles of Concurrent and Distributed Programming", ch. 16, Prentice Hall, 1990, ISBN 0-13-711821-X, page 164
3. Martin, James (1965). *Programming Real-time Computer Systems* (https://archive.org/details/programmingrealt0000mart). Englewood Cliffs, NJ: Prentice-Hall Inc. p. 4 (https://archive.org/details/programmingrealt0000mart/page/4). ISBN 978-0-13-730507-0.
4. Kant, Krishna (May 2010). *Computer-Based Industrial Control* (https://books.google.com/books?id=3714jIryozYC&pg=PA356). PHI Learning. p. 356. ISBN 9788120339880. Retrieved 2015-01-17.
5. Shin, Kang G.; Ramanathan, Parameswaran (Jan 1994). "Real-time computing: a new discipline of computer science and engineering" (http://kabru.eecs.umich.edu/papers/publications/1994/ramanathan-shin-ieee-proceedings.pdf) (PDF). *Proceedings of the IEEE*. **82** (1): 6–24. CiteSeerX 10.1.1.252.3947 (https://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.252.3947). doi:10.1109/5.259423 (https://doi.org/10.1109%2F5.259423). ISSN 0018-9219 (https://www.worldcat.org/issn/0018-9219).
6. Kopetz, Hermann ; *Real-Time Systems: Design Principles for Distributed Embedded Applications*, Kluwer Academic Publishers, 1997
7. Liu, Chang L.; and Layland, James W.; "Scheduling Algorithms for Multiprogramming in a Hard Real-time Environment", *Journal of the ACM*, 20(1):46-61, January 1973, http://citeseer.ist.psu.edu/liu73scheduling.html
8. Menychtas, Andreas; Kyriazis, Dimosthenis; Tserpes, Konstantinos (July 2009). "Real-time reconfiguration for guaranteeing QoS provisioning levels in Grid environments". *Future Generation Computer Systems*. **25** (7): 779–784. doi:10.1016/j.future.2008.11.001 (https://doi.org/10.1016%2Fj.future.2008.11.001).
9. Kuo, Sen M.; Lee, Bob H.; and Tian, Wenshun; "Real-Time Digital Signal Processing: Implementations and Applications", Wiley, 2006, ISBN 0-470-01495-4, Section 1.3.4: *Real-Time Constraints* (http://media.wiley.com/product_data/excerpt/54/04700149/0470014954.pdf).
10. Kudrle, Sara; Proulx, Michel; Carrieres, Pascal; Lopez, Marco; et al. (July 2011). "Fingerprinting for Solving A/V Synchronization Issues within Broadcast Environments". *SMPTE Motion Imaging Journal*. **120** (5): 36–46. doi:10.5594/j18059XY (https://doi.org/10.5594%2Fj18059XY). "Appropriate A/V sync limits have been established and the range that is considered acceptable for film is +/- 22 ms. The range for video, according to the ATSC, is up to 15 ms lead time and about 45 ms lag time"

11. Stankovic, John (1988), "Misconceptions about real-time computing: a serious problem for next-generation systems", *Computer*, IEEE Computer Society, **21** (10), p. 11, doi:10.1109/2.7053 (https://doi.org/10.1109%2F2.7053), S2CID 13884580 (https://api.semanticscholar.org/CorpusID:13884580)
12. "Federal Standard 1037C: Glossary of Telecommunications Terms" (http://www.its.bldrdoc.gov/fs-1037/fs-1037c.htm). Its.bldrdoc.gov. Retrieved 2014-04-26.

Further reading

- Burns, Alan; Wellings, Andy (2009), *Real-Time Systems and Programming Languages* (4th ed.), Addison-Wesley, ISBN 978-0-321-41745-9
- Buttazzo, Giorgio (2011), *Hard Real-Time Computing Systems: Predictable Scheduling Algorithms and Applications* (https://books.google.com/books?id=h6q-e4Q_rzgC&q=%22real-time%22), New York, NY: Springer, ISBN 9781461406761.
- Liu, Jane W. S. (2000), *Real-time systems*, Upper Saddle River, NJ: Prentice Hall.
- The International Journal of Time-Critical Computing Systems (https://www.springer.com/journal/11241)

External links

- IEEE Technical Committee on Real-Time Systems (http://sites.ieee.org/tcrts/)
- Euromicro Technical Committee on Real-time Systems (http://www.ecrts.org)
- The What, Where and Why of Real-Time Simulation (http://www.opal-rt.com/technical-document/what-where-and-why-real-time-simulation)
- "DESIGN OF A REAL-TIME PROGRAMMING SYSTEM" (https://archive.org/details/bitsavers_computersA_7555012/page/n25?q=%22DESIGN+OF+A+REAL-TIME+PROGRAMMING+SYSTEM%22). *Computers and Automation*. **XII** (9): 26–34. Sep 1963. "[...] set of notes which will hopefully point up problem areas which should be considered in real time design."