



REAL-TIME SYSTEMS

Graham Berridge
TESSELLA SUPPORT SERVICES PLC

Issue V2.R1.M0
June 2005



1. Introduction

This technical supplement looks at what types of systems could be defined as 'Real-Time' and then discusses the various issues and options in choosing and developing such a system. Also highlighted are the difference between general-purpose Operating Systems (OS) and specialist Real Time Operating Systems (RTOS).

2. What is a Real-Time System?

The primary difference between 'ordinary' and 'real-time' systems is the latter's design constraints on its response to external events. These constraints define that some response is expected within a known time, and that this can be guaranteed. In theory any system, by providing adequate processing speed, could be made to fit this constraint, but the other characteristic of 'real-time' systems is that they are usually constrained in various other ways that could preclude this (such as limited power consumption).

So we have defined a system where events must give rise to controls or actions within given times. The actual response time of a system is known as its 'latency' and this, together with the processing time, must obviously be smaller than any desired response time. Any failures to act in this time can be catastrophic; for example not closing an injection valve in a combustion engine in time leaving it to be hit by the piston. This does not mean that the system must be the fastest possible, as this may not be possible, it just must be 'fast-enough'.

When the time constraints are strict and the processing response is rigidly defined, then the system can be termed 'hard' real-time. Systems where the constraints are easily met are then termed 'soft' and, although there is certainly no clear distinction between the two extremes, hardness depends heavily on the chosen platform and the implementation. As a guideline 'hard' real-time systems are characterized by response times of the order of microseconds, while 'soft' systems tend to have milliseconds or longer to respond.

3. Other Constraints on Real-Time Systems

As already indicated above, real-time systems are usually constrained in several ways in addition to response time. One is obviously cost (you cannot put a £100 processor in a device you want to sell at £5), and system power consumption has already been mentioned (you expect hours of talk time on a mobile phone, not minutes). Others are physical size, reliability and robustness (again consider a mobile phone). Some of these combine to affect the amount of memory the

system may have.

Additionally, there is the hardware it is connected to. The system takes data and signals from some input devices, processes it, and transmits data and control signals to some output devices. These devices are typically custom hardware tailored to the specific application, and the real-time system is going to be directly connected to them, unlike the standard interfaces you would use with a general-purpose system (like USB or parallel ports). The usage will also be different with a real-time system likely to use a measure-process-control model compared with transmitting commands 'blind' in a general-purpose system.

Memory constrains the real-time system in two ways. Firstly, the operating system and application code have to be stored and the total available space may be restricted (e.g. some systems use internal Flash memory that is of limited size, rather than a disk device). Secondly, there must be sufficient working memory for the application data and operating system overhead including stacks and internal buffers. Real-time systems usually approach the first problem by using specialized operating systems that allow unnecessary parts of the OS to be omitted (e.g. the floating point libraries or networking code), to reduce size or by avoiding the use of an OS altogether and having a simple event handler loop. In these cases, Windows NT or even XP Embedded would be poor choices since even the 'modular' XP Embedded can only be cut down to a minimum of 5Mb.

Processing speed is another subtle area of constraints. A given processor may be adequate in many areas but not have hardware floating point acceleration. In applications where complex numeric calculations are required, this could put a significant performance penalty since software mathematics libraries are inherently slow. A solution here might be to save the memory and use fixed-point maths, using integer mathematics support which is much faster but requires careful programming to ensure that the required accuracy and magnitude of numbers (at all steps in a calculation) is preserved.

One final area of interest in the constraints of real-time systems is an ability to be updated easily and possibly remotely. Perhaps the most famous example of this is the Mars Sojourner with its roving robot. The robot was suffering from a problem where one piece of control software was waiting on another which never finished – this was diagnosed on Earth using replica hardware and the required software update was loaded dynamically onto the robot on Mars. This is an

extreme case, but maintenance access must be considered an issue if the system is not easily accessible or must remain active during such an update (rebooting may not be an option).

4. The System Hardware

In referring to a real-time system we have been discussing some 'target' hardware consisting of the special hardware with a controlling processor and the required support chips. It could be a custom system, an off-the-shelf Single Board Computer (SBC) (which itself could be a standard PC in a compact form), a normal or rugged PC, or even a Programmable Logic Array of some sort.

The application-specific hardware attached to the processor could include the following:

- ❑ a full keyboard, but more likely a keypad with a few keys: there might only be a reset button
- ❑ a display of some sort like a VGA panel or a custom pattern LCD: there might only be a power LED
- ❑ sensors for digital and/or analogue signals from the application environment which includes serial and parallel data streams
- ❑ transducers for digital and/or analogue signals to the application environment.

Table 1 lists a few types of system that could be used for real-time processing including some data related to the various constraints mentioned above (entries are guideline only). Note that the spread of cost is more a measure of how much of a commodity product the item is than a measure of flexibility (the rack based VME/CompactPCI systems are highly specialized and low-volume production items).

Description	Processor (e.g.)	Speed (MHz)	Memory (Mb)	Power (W)	Storage (Mb)	IO Devices & bus type	Cost (£)
Generic PC	Pentium III	800	512	200	40,000	PCI & ISA bus. RS232, parallel & USB ports. SVGA video, mouse, keyboard. IDE & SCSI disks, network.	400
SBC (PC104 format)	80486 SX	25	16	15	2	RS232 port. IDE disks, network. VGA video, mouse, keyboard. PC104 units stack together: digital/analog IO, disks. ISA bus.	80
Rack-mounted cards (e.g. VME or CompactPCI bus backplane)	SPARC, PowerPC or Pentium	1000	256 onboard + shared over bus	50	0 to massive shared over bus	VME/PCI bus between cards. PCI/processor bus on-board/ RS232 & network. Many other devices over the backplane.	3000
Embedded controller	H8, 8051	16	64kb	0.05	0	4x 10bit ADC, serial ports, digital IO, counter/timers.	25
PDA	StrongARM	200	32	0.15	Uses memory	RS232, USB ports. PCCard slot.	150
Digital Signal Processor	SHARC DSP, TMS320	50 2 proc. units	0.5 dual-ported	0.5	0	Serial ports and data streams.	20

Table 1

Each device attached to the processor requires a software interface; in the simplest case this defines the format and timing of data read and written from registers, while in a higher-level case a device driver would be used to protect the application from the device. Device drivers may be provided with the OS, if the device is common, or can be written specially if the device follows the normal rules for device drivers on the system. Failing a suitable model in the OS, a custom interface within the application will be required.

It is also worth mentioning that some real-time tasks may suit a Digital-Signal-Processor (DSP) architecture better than the more general hardware illustrated above. DSP systems are optimized for data-stream capture, real-time transformation using optimized floating-point processing and filtering and data-stream output. Applications include speech and video processing/compression or radar/sonar imaging. Although seen as lower level than embedded processors, their high mathematics performance coupled with a similar programming model to the other systems makes them very useful, even as part of a larger system.

5. Real-Time Software

Software for real-time systems is commonly developed on a 'host' computer which has development tools and compilers that can create suitable code images for the 'target'. Since in many cases the development machine is a different sort of computer from the target platform, a 'cross-compiler' (a compiler that runs on the host to create output for a different OS or platform) is used to generate this image. In rare cases the target has the facilities to support development on board, for example using the OS/9 operating system, much as for developing for a general-purpose OS like Windows, but this may cause problems due to the memory and storage required for the tools restricting the system during testing. The most common tools for cross-platform development are the GNU compilers, available for most platforms and target hardware. Most hardware vendors will have a compiler set for their own processors as well, often derived from the GNU tools.

Once created, the image has to be transferred to the target (unless it has been created on it). This can be done in many ways depending on the OS and the hardware. Often there is network support and the image can be transferred and loaded onto the target using FTP or NFS from a command mode or a minimal loader system running on the target. In other cases a serial connection can be used to transfer the image using a special mode of the target (e.g. for Flash memory programming). The host can have a direct hardware connection to the target and supply the memory image directly using an In Circuit Emulator (ICE). In the most detached cases the image might be written to programmable memory chips (EPROMs) which are then physically inserted into the target.

Real-time software is most commonly written in the C programming language. This is due to the way the language maps onto machine level instructions, making it fast (if properly designed!) and easier to follow at the machine level. This is helped by the ubiquity of the GNU compiler and its support for most

processors. Assembly language is still used for performance and some hardware device access. The Object-Oriented language C++ is gaining popularity as the benefits of enhanced design are seen to outweigh the possible overhead of supporting the more complex run-time environment. Java is beginning to be used (look at the latest mobile phones), due to its ease of portability between different hardware and the possibility of having optimized versions of the Java Virtual Machine (JVM) based on the J2ME standard (a platform with no user interface need not implement the graphical aspects of the JVM and can handle the memory protection more efficiently).

During development, high level debugging tools are useful, and some systems allow full symbolic access on the target with performance profiling via an environment on the host. The use of an ICE allows the host to control the target and perform high-level debugging. Support is usually required for both of these within the target's operating system. In other cases there may be an assembler level debugging facility on the target, or the developer may need to resort to a variation of printing out progress messages; this can be difficult without a text display or serial port! The only other option that may be available in these cases is if the host tools provide some form of software emulation of the target.

6. The Operating System

It is a great help to software developers to have a rich Application Programming Interface (API) to provide facilities, libraries and constructs to speed development. This is one purpose of the Operating System (OS). The other is to provide services to the application. A real-time OS (RTOS or 'kernel') typically provides the following services through the API:

- ❑ **Multiple threads of execution:** an RTOS may provide light-weight threads or heavier-weight processes (or both), often referred to as 'tasks'. Threads are characterized by having a small overhead on switching between threads while data is global to all threads. Processes have a higher cost of switching but data is not visible outside of the process. The RTOS will manage the scheduling of threads/processes through the use of priorities which the application can adjust dynamically.
- ❑ **Pre-emptive scheduling:** the kernel selects the task to run that has the highest priority and that is not waiting for some service. If more than one task with this priority is able to run, they are usually given a fixed slice of processing time in turn (known as 'round-robin' scheduling). This method allows real-time systems to be 'deterministic', meaning that

given a known set of inputs and priorities, the actual processing and scheduling performed can be predicted; general-purpose systems use dynamic priority boosts if a process has not had a chance to run, resulting in chaotic scheduling.

- ❑ **Synchronization services:** these services allow tasks to communicate events to each other.
A Mutex (mutual exclusion) allows only one task to own it at a time – any other tasks are made to wait (they are ‘blocked’) until the owning task releases it; these can be used to protect access to devices or data.
A Semaphore is a simple signal that allows one task to indicate to another that something has happened, blocking the second until it sets or increments the service (similar to an Event).
A Critical Section is like a Mutex that forces only one task to run a piece of code at a time, without risk of being pre-empted.
- ❑ **Other intertask communications:** some sort of message queue is often provided to allow fixed sized packets of data to be sent between tasks (a ‘pipe’). The length of the queue and the size of the items in it are fixed when it is created. Queues can be FIFO (first in, first out) or priority based (higher priority messages get pushed higher up the queue), and also act as synchronization services.
- ❑ **Memory management:** in the general-purpose computer case, memory management usually means providing virtual memory (using disk to extend the actual available memory) – this may well not be available and the real-time system needs to be much more careful with its usage of memory. The memory services in an RTOS often include support for multiple areas of memory for allocations where each area is used for a different range of sizes. This reduces memory fragmentation since the gaps between blocks will be of the size of the next block to be allocated.
- ❑ **Interrupt support:** a means of mapping various external events to the code required to service the action is required. These ‘interrupt handlers’ work in a slightly different way to normal task code in that there are restrictions on the kernel actions they can perform: no waiting on synchronization is allowed, but a semaphore could be set to release a waiting task. Handlers need to be as short as possible so that other interrupt handlers are not postponed while another is being processed (interrupts remain active until explicitly cleared). Of course, if the generation of interrupts is more frequent than can be handled then events will be lost – the system must be fast enough.

- ❑ **Posix support:** a standard set of APIs that allow for easier porting of applications between RTOSs (and UNIX-like OSs).

One further important feature that should exist in the synchronization services is 'priority inheritance'. This is required to prevent 'priority inversion', which is the failure on the Mars Sojourner robot mentioned before. The problem was this: a high priority task was waiting on a Mutex owned by a low priority service, but sometimes there was a mid-priority task able to run – the low priority task never got to run and never released the Mutex so the high priority task never ran. The way this is prevented is to allow the owning task to be boosted to the priority of the waiting task until it releases the synchronization service, i.e. the owning task inherits the waiting task's priority.

Applications written to make use of tasks must ensure that application interface routines (routines communicating between tasks) are 're-entrant', meaning that a routine has no side-effects. This usually means not changing global or device data and using synchronization services to prevent contention if such access is required. This also applies to any common libraries used: there are often single- and multi-threaded versions of libraries available, the former being smaller and faster but without the re-entrancy and task protection.

Selecting an operating system for a real-time system is as important as the rest of the system design. As has already been described, a real-time system has to conform to the constraints placed on it, and the RTOS is the prime consideration, together with the actual hardware, in meeting those constraints. If the interrupt handling latency is too long, for example, then a different RTOS and/or hardware specification must be chosen.

There are many issues to consider. For example, it might be thought that a general-purpose OS, like Windows 2000, might suit the application; after all, the development tools are readily purchased and there is a lot of expertise available which helps reduce the development costs. However, after consideration it might be realized that the hardware will not support suitably large disk drives or have sufficient memory to be effective in the limited cost. Even if the platform is adequate other requirements may not be met:

- ❑ **Servicing of frequent interrupts:** the processor could become choked as a result of poor task switching times and OS overhead like paging of virtual memory (tasks over which the developers have no control).
- ❑ **Determinism:** the interrupt response and the running of tasks to

service the device may need to be absolutely predictable: a feature not available with OS-controlled dynamic priorities. The interrupt latency will also not be very well defined.

- ❑ **Application design:** the facilities offered by an RTOS allow a software design that is highly modular, relating data transformations to tasks quite easily. This is not a common model on general-purpose systems where inter-task communication is often difficult to program and not efficient (such as including process security features requiring lengthy validation by the OS).
- ❑ **Hardware support:** the creation of drivers for custom hardware can be a difficult task since not only must the application interface be written and tested but so too must the OS interface. Additionally, drivers for common specialized devices may not be available for a general-purpose OS even though they may be available for a specialized RTOS.
- ❑ **Cost:** the cost of an OS can greatly affect the final cost of a product. In most cases the OS has an up-front developer license cost but there may also be a per-item royalty cost as well that could make the product very expensive.
- ❑ **Portability:** the types of hardware used for real-time systems tend to be either off-the-shelf standard units or custom built boards; either can result in hardware change due to external influences (e.g. if the current processor becomes unavailable or too expensive). The resulting re-evaluation can lead to the platform being changed, so portability of the application software and the availability of the existing OS on a wide selection of hardware can allow product and turn-around cost to be minimized.

This is not to say that the generic-operating system will not be suitable, but that it may not be the best choice. The following table lists some of the currently available RTOSs. It is certainly not an exhaustive list!

RTOS	Supplier	Hosts	Target Processors	Notes
INTEGRITY	Green Hills	Windows, Unix	x86, PowerPC, ARM, MIPS, XScale	Robust systems
ThreadX	Green Hills	Windows, Unix	68k, PowerPC, i960, ARM, MIPS, SH, SPARC	Lightweight & responsive RTOS
VxWorks	WindRiver	Windows, Solaris, Unix	x86, 68k, PowerPC, i960, ARM, MIPS, SPARC, ...	Proven for industrial and military systems. Uses threads
pSOSystem	WindRiver	Windows, Solaris, Unix	x86, 68k, PowerPC, i960, ARM, MIPS	Lightweight & responsive RTOS
LynxOX	LynuxWorks	Linux	x86, PowerPC, MIPS	Real-time version of Linux. Full POSIX conformance
BlueCat	LynuxWorks	Linux	x86, PowerPC, ARM, MIPS, XScale, IA-32	Real-time version of Linux. Uses kernel 2.6
RTLlinux	FSMLabs	Linux	64-x86, x86, PowerPC, ARM, MIPS	Hard real-time POSIX OS.
CMX RTOS	CMX	Windows	Most 8, 16, 32 & 64 bit processors	Very small footprint
QNX neutrino	QNX	Windows, Solaris, QNX	x86, PowerPC, ARM, StrongARM, XScale, MIPS, SH-4	Highly flexible, small footprint
Windows CE	Microsoft	Windows	x86, StrongArm, MIPS, SH	WinCE 2.0 was not an RTOS. WinCE 3.0 is closer but with poor latencies
DSP/BIOS	Texas Instruments	Windows	TMS320 based DSPs	Full kernel special support for DSP processing units

Table 2

General-purpose operating systems that have been used for real-time systems include OpenVMS (from Compaq and based on VAX/VMS), Solaris (from SUN), Windows NT based Oses (from Microsoft) and Linux. However, the processor requirements for these is naturally much higher than for an RTOS, to perform acceptably as a real-time system some third-party add-ins may well be required (especially for Windows NT, 2000, XP and XP Embedded), and the range of available processors is quite small.

7. Conclusions

The selection and design of hardware and software for a real-time system depends very heavily on what it is required to do and the facilities required. There are many constraints on the solution, ranging from the time domain to the operating environment. Specialist real-time operating systems exist targeting highly constrained applications that provide the predictable response times, long product life-time and robustness required. If access to standard products like Microsoft Office is required then a general-purpose OS must be considered. In the event that both aspects are required then 3rd party add-ons (e.g. RTX for Windows XP from VenturCom) can be used to provide the deterministic behaviour of a true RTOS on, say, Windows.

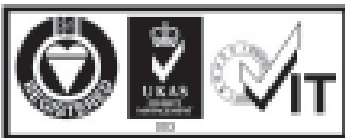
Every real-time system is different and the solution found will be unique. This supplement has shown the range of options available and the constraints to consider as a broad introduction for locating an optimal solution.

Tessella - providing innovative software solutions to scientific, technical and engineering problems

Tessella specializes in the application of innovative software solutions to scientific, technical and engineering problems. Our services cover software design and development, IT consultancy, infrastructure support and project management.

Other Technical Supplements published by Tessella include:

- Active Server Pages
- Archiving of Electronic Information
- Automated GUI Testing
- Bayesian Statistics
- Beowulf Clusters
- Beyond LIMS
- C++
- Choosing and Using a LIMS
- COM
- Computational Fluid Dynamics
- Computer Image Processing
- Decision Support Systems
- Development for the Mobile Platform
- Digital Preservation Practical Experiences
- e-GIF
- Electronic Data Capture
- Electronic Lab Notebooks
- Evolutionary Computing
- Excel
- Extending the Life of Software
- FDA21 CFR Part 11
- Formulation
- FORTRAN 90
- Grid Computing
- High Throughput Experimentation
- High Throughput Screening
- Instrumentation
- Integrated Lab Systems
- J2EE
- Java
- Linux
- Microsoft .NET
- n-tier Architecture
- Object Oriented Programming
- Open Source and Free Software
- Pocket PC
- Portable GUI Development
- Real Time Systems
- Regression Testing
- Security and the Internet
- Simulation
- Soft Computing
- Software Development Cycle
- Software Documentation
- Software Portability
- Software Re-engineering
- Software Specification
- SQL
- UNIX Inter-Process Comms
- UNIX System Performance
- Web Services
- Windows 2000 Services
- Workflow Systems
- XML
- X Windows



Certificate No. FM 22778



INVESTOR IN PEOPLE

Tessella Support Services plc

3 Vineyard Chambers, Abingdon, Oxon, OX14 3PX, England

Tel: (+44) (0) 1235 555511 Fax: (+44) (0) 1235 553301

info@tessella.com

www.tessella.com