You can **lean on me ...**

What relevance does an OS designed in the 1980s have to embedded systems today? By **Vanessa Knivett**.

Rooted in the early 1980s – specifically, the Tandy Colour Computer 3 – the OS-9 operating system is proving to be something of a comeback kid. With a large installed base, it continues to gain design wins today. So why is something that was originally developed as a platform on which to run a basic language interpreter more than 20 years ago, continuing to gain followers?

Ric Yeates, senior software architect on RadiSys' OS-9 project, attributes OS-9's popularity to several factors. Firstly he suggests: "It was used to support the Motorola 6809 processor (which featured in the popular Tandy computer) and it was a logical progression to convert this run time environment to a full OS. So that's where it got its size and performance characteristics."

Although the 6809 was a 2MHz processor, OS-9 enabled it to scale to work with one user or tens of users – as developer Microware did internally. Recounts Yeates: "Customers were using OS-9 in one embedded system out in the field. However, we were using it in house to run our development and sales/marketing operations. Unlike previous OS', we didn't have to reboot OS-9 several times a day." OS-9 can be scaled from a completely diskless environment without I/O, to a full system supporting serial, disk, internet and more, unlike Unix' structure, which had provided inspiration for OS-9's developers.

Its real time, multiuser, multitasking characteristics are down to the modular architecture to which OS-9 subscribes, according to Yeates. New devices can be added to an OS-9 system simply by writing device drivers, with I/O devices treated as files. Applications and drivers can be dynamically loaded and unloaded during development or after deployment without a reboot.

Allen Huffman, a software engineer at RadiSys, explains the OS-9 structure more fully: "When OS-9 was designed in 1980, the concept was that code would be living in rom and could be mapped anywhere. The OS-9 module format was created to allow for position independent code and to let components be swapped in or out. A hardware device might contain OS-9 drivers in rom, which would then appear and be available to the system – a concept that allowed the Philips CD-i (Compact Disc Interactive) systems to plug in Mpeg decoders and allow OS-9 (it was called CD-RTOS) to have Mpeg drivers for playing VideoCDs."

As shown in figure 1, OS-9 consists of a group of modules, organised in a multilevel approach. The system modules comprise:

- *kernel*, which contains the code for process scheduling, semaphors, events and memory allocation
- *init*, a data block with kernel defaults that contains the name of the initial launch process, memory map, default table sizes (such as path table, event table)
- *ioman*, which can be used to give access to a Unix like unified I/O system for files, serial streams and tape devices
- *clock*, a software handler for the specific real time clock hardware.

Other system modules include the ticker (heartbeat), ssm (mmu manager), cache, and kernel extensions to handle

Illustration: Elly Walton

"OS-9 shines in applications where high performance, high reliability and scaleability are needed."

Mike Lottridge, **Radisys**

external hardware like the MMU or IRQ controllers.

Underneath the system modules and ioman are the file managers. The file manager module consists of a random block file manager which takes raw sectors from a disk device driver, then navigates the OS-9 disk structure. Adds Huffman: "The same drivers can be used with a pc file manager to navigate a pc dos style file system. So, it is the file manager that does all the heavy lifting to make the data useful. This really helped back in the 1980s and 1990s, when memory was expensive." Another file manager, SCF (sequential character file manager), manages streams of bytes.

Underneath each file manager are device driver modules, which provide raw device access. Then comes the device descriptor modules – these contain the configuration data used by a device driver. Explains Huffman: "A driver may know how to talk to a 16550 serial chip, but the descriptor defines an instance of that chip in memory (hardware base address, IRQ vector and baud rate). This lets one device driver run dozens of instances, each with their own device descriptor describing where that instance is in memory."

The application modules talk to the kernel and share data modules, which are user defined. These can be used in rom to store data (for a file less system), or can be dynamically created and shared (shared memory).

The evolution of OS-9

Over the years, multiuser applications of OS-9 dwindled, whilst continuing applications tended to use it as a development platform, rather than a host. The OS itself changed too. Explaining the three main stages of OS-9's evolution, Yeates said: "The first evolution of OS-9 was that it was ported to the 68000 family, bringing it to a higher performance processor and 16 and 32bit capabilities." This afforded developers the opportunity to add more functionality (this period saw it used in a variety of industrial and commercial applications, including Philips' CD-I).

Yeates sees the second major evolution as being in the early 1990s, when a C based version was created: "That was the largest change for OS-9 because that allowed it to be run on any processor for which we had a C compiler." Currently a wide range of processors is supported, including 68xxx, PPC, X86, Intel SARM/IXP, MIPS, SPARC, and Hitachi SH.

The final generational change was the introduction of threads. He explains: "OS-9 has always been process based, so when processors encounter a problem, the process can terminate it without endangering the rest of the OS. Most competing OS' were monolithic, thread based – so a catastrophic problem in one area could bring the whole system down." Microware added multiple threads of execution within a single process. The benefit, explains Yeates, is that when writing an application, it is isolated from the main OS so any errors introduced by the designer don't impact the system. Microware used the Posix standard to create threads, so any existing Posix application can be ported to OS-9.

Explaining OS-9's relevance today, Mike Lottridge, Radisys' software product line manager, says: "OS-9 shines in applications where high performance, high reliability and scaleability are needed. Areas it is being used in today include industrial automation, car navigation, test equipment, imaging and traffic control systems."

Alluding to competitor OS' such as embedded Linux, Lottridge makes the case for going against the open source tide: "RadiSys has a fairly substantial hardware business, of course, so it can supply hardware, the OS, compiler, development environment – all the tools from one vendor, for at one time, as many as 12 processors. Customers also value the way we treat the source code – we work hard to keep things backwards compatible as customers typically have very long life systems."

Figure 1: OS-9 module organisation