Experiences with a COTS Real Time Operating System for a Satellite On Board Computer

A. F. Dachs and H. A. B. Tiggeler

Surrey Satellite Technology Limited
Surrey Space Centre
University of Surrey
Guildford
Surrey
GU2 5XH
UK
Tel: +(44) 1483 259278
Fax: +(44) 1483 259503
Email: A. Dachs@ee.surrey.ac.uk, H.Tiggeler@ee.surrey.ac.uk
http://www.sstl.co.uk

Abstract

Surrey Space Centre (SSC) micro-, mini- and nanosatellites depend heavily on their on-board computers for performing tasks such as attitude control, payload support, communication and housekeeping. Recent improvements in the resolution of imaging systems and higher bandwidth communication links have resulted in further demands on the On Board Computers and software.

This paper describes the experience gained and lessons learned from porting a popular Commercial Off The Shelf (COTS) operating systems to one of SSTLs on-board computers. The operating system described is QNX ® Real Time Operating system [1] developed by QNX Software Systems Ltd.

Background

UoSAT-1, the first microsatellite built at the University of Surrey was launched in 1981. Since then a total of fourteen microsatellites in the 50 to 100kg class and one 350kg minisatellite have been launched.

Over this time the size and mass of the Surrey microsatellites has remained relatively constant but the capability increased significantly. Since the On Board Computer (OBC) is responsible for monitoring, commanding and collecting data from the bus systems and payloads, it's role has also increased.

Accordingly the hardware and software have been modified and improved while maintaining as much heritage from previous systems as practical. Intel microprocessors were used in the On Board Computers from an early stage, starting with the Intel 80186 on UoSAT-3. A 386 based OBC has been used on the more recent TMSAT, FASAT and UoSAT-12 satellites.

The complexity and functionality of these more recent satellites has shown a marked increase compared to earlier microsatellites. One measure of complexity is the number of processors per microsatellite. Figure 1 shows this trend for the UoSAT class of satellites. The phased introduction of a distributed Telemetry and Telecommand system on FASAT Bravo and TMSAT contributed the majority of the new processors on the microsatellite bus. UoSAT-12 is our first minisatellite and is also our first satellite using a fully distributed Telecommand and Telemetry system.

The amount of data stored on the satellite has shown a similar increase (see Figure 2), mainly due to increased image resolution. Storing and manipulating this amount of data has implications for the on board networks and computers.

Future satellites will require computers capable of sustaining higher data rates either by improving single OBCs or by sharing the load between multiple OBCs.

The development of the minisatellite platform has provided an opportunity to use multiple On-Board Computers connected over a local area network. However new missions have been proposed where power consumption, mass and volume are the key drivers.

These diverse requirements lead to multiple hardware solutions with serious implications for the system software. Software reuse across the different missions is essential to keep the costs under control. In order to reuse software on different hardware some form of

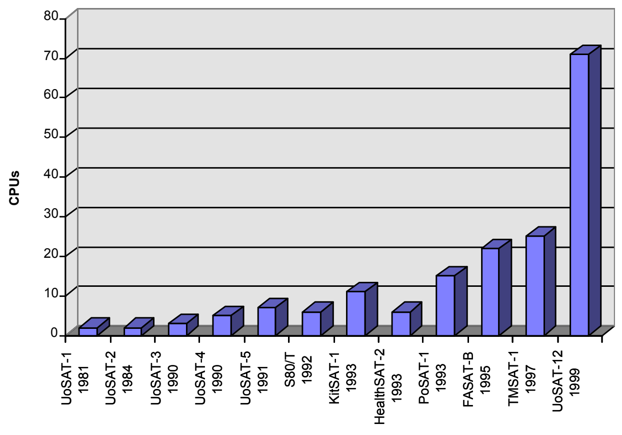

Figure 1. The number of processors per satellite

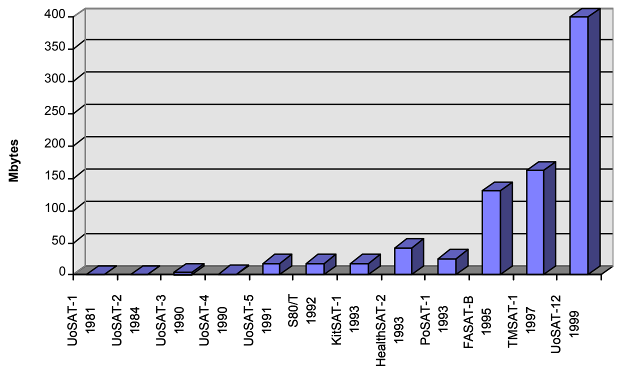

Figure 2. Memory on each satellite

abstraction is required. Apart from providing multitasking and some basic services an operating system is also a layer between the hardware and software. It can provide a common interface for application software.

An operating system that supports multiple hardware platforms should be able to run the same, or similar, software on all of the platforms. If the operating systems supports the POSIX interface standards then the software should be portable to other operating systems with minimum effort. The "Portable Operating System Interface for Computer Environments" (POSIX) standards were established by the IEEE and are still under development [2].

QNX is a POSIX.1 certified microkernel operating system [1] developed by QNX Software Systems Ltd (QSSL). QNX 4.x is specifically intended to run on PC hardware although the newer Neutrino product has multi-platform support. This paper describes the porting of QNX 4.24 to the OBC386.

Operating System Selection

The current operating system is called SCOS, which stands for "Space Craft Operating System", and was specially developed for the amateur satellite community. SCOS was originally derived from the Quandron QCF operating system and was adapted for micro-satellite use by BekTek in the USA in 1986. Since that time SCOS has been used on more than 26 micro and mini-satellites and is still the preferred choice for 186 based on-board computers. However the i386 processor provides additional multitasking and memory management functions which enhance individual task integrity and protection. These functions are currently not supported by SCOS.

The task of selecting a new operating system was not a trivial one but the choice could be narrowed down by applying some criteria specific to our application. The ones we identified were:

- Availability of development tools, compilers and debuggers hosted on a PC
- Small, efficient kernel
- Proven, stable kernel
- Low Interrupt Latency
- Availability of drivers for standard hardware (such as Ethernet)
- Strong customer support (newsgroup, www, freelance programmers)
- Fast context switch and inter process communication
- Fault tolerance
- POSIX compliance
- Support for existing 386EX platform and an upgrade path
- Multi-Processor support
- Multi-Platform support

In this instance we were not restricted to selecting an operating system that was space qualified or approved. The use of a proven commercial operating system is consistent with SSTL's use of commercial off the shelf components in its spacecraft.

QNX was selected as it met most of our requirements. It is a well developed commercially available real time operating system for the i386 (recently support has been added for the PowerPC and MIPS family of processors). It runs in protected mode so it has a full 32 bit address space and takes advantage of the memory protection mechanism in the i386. It is POSIX compliant and similar to UNIX in many respects but is small and has low interrupt latencies making it suitable for embedded applications. In addition to the kernel and drivers, QNX comes with many utilities commonly found on UNIX systems. QNX and the Watcom C/C++ compiler, which is well regarded for reliability and code optimisation, can be used on a standard desktop PC for development.

Implementation

Once the operating system was selected the porting process was implemented in four stages. The first stage was to build a customised kernel with a basic set of functions. The second stage was to write device drivers for the application specific hardware and to write support utilities. The final stage was to port the existing SCOS applications to the QNX environment. Figure 3 is an overview of the tasks that need to run on the flight OBC.

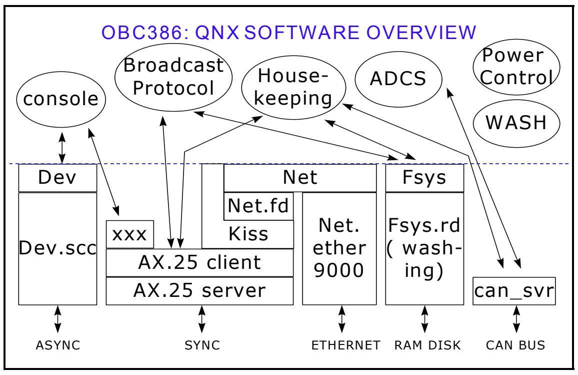

Figure 3: Overview of OBC software

Customisation

The process of porting the kernel to the OBC386 was relatively straightforward. This was due to the fact that QNX is primarily designed to run on IBM PC compatible computers and the 386EX processor used on the OBC386 contains PC based peripherals. The embedded toolkit was used which contained a sample configuration for the 386EX processor. The sample could be modified to work on the OBC.

Since most embedded systems do not have a hard disk or floppy drive QNX provides utilities to create a boot image [3]. This image can be programmed into non-volatile memory or loaded directly into memory and executed. A boot strap loader was developed to perform the latter task.

QNX makes a number of standard PC BIOS calls to communicate with the outside world and to determine the hardware configuration. The video display and keyboard input calls were re-directed to one of the 386EX built-in serial ports. Once the image is loaded into memory it expands itself and starts the operating system.

Development Environment

A PC is used as the host for development purposes and many applications can be written without the embedded target present. Standard QNX and Watcom C are installed on the PC.

A serial port is used to bootload the OBC and later for console I/O. The ethernet connection is used to copy files across the network and is also useful for recovery if an errant program locks the console.

Device Drivers

Device drivers provide a high level interface to hardware that is not directly supported in the operating system kernel. QNX has a microkernel architecture so very little hardware support is built into the kernel and most hardware is handled by device drivers. Some of these are provided as standard with the operating system but drivers for any non-standard hardware had to be written.

The microkernel architecture of QNX means that device drivers can be written as user tasks and started or stopped at will. Like user tasks, device drivers run in their own memory space so the kernel is protected from faults in the driver which is particularly useful during development. However, a device driver needs to access real hardware, attach to interrupt vectors and access physical memory so it requires a high privilege level. Efficiency and latency are also critical. These types of functions are not necessarily portable but it is vital that the operating system supports their use. QNX provides access to these functions through a mixture of POSIX and QNX-specific mechanisms.

The custom drivers were written using a client/ server model. Messages are passed between clients and servers with the QNX synchronous messaging IPC mechanism. This model avoids some overhead compared with a full POSIX I/O manager implementation at the expense of portability. Since efficiency was important and QNX specific calls were required for the hardware interfacing it was felt that the trade-off was justified.

The SMC91C94 ethernet chip was chosen because of the simple hardware interface and because it was supported by QNX. The standard QNX driver could be used directly with the QNX network manager, Net. Therefore the ethernet chip could be replaced with any other supported chip without affecting the application software.

Net provides a hook for sending and receiving raw packets so a custom network server was written to coexist with QSSL's Net manager. Net handles QNX specific packets and the custom server handles file transfers from the imaging system or SCOS OBCs.

Support Utilities

A time consuming part of the porting process was writing satellite specific support utilities such as the "memory wash routine", error detection and correction for the ramdisk and the power management task. These functions were built into the SCOS kernel but were not present in QNX.

Program Memory Wash
The program memory on the OC386 is protected by a hardware Triple Modular Redundant scheme (TMR). Radiation induced Single Event Upsets (SEU) [4] can cause memory bits to change state but the hardware corrects any single SEU errors in any bit. If errors are allowed to accumulate the error correction will fail so a periodic memory "wash" is required. Memory washing, or error scrubbing, is a simple process of reading and writing back each memory location.

Gaining access to all of the physical program memory is not straightforward when in protected mode as any user task attempting to read or write memory outside its allocated space will cause a protection fault. The source code for the QNX kernel is not easily available so there was no opportunity to build this directly onto the kernel. Modifications at this level would have also required a good working knowledge of the kernel.

The above problems were avoided by using the 386EX DMA controller to do memory to memory transfers. The DMA controller on the 386EX could be used because it has an atomic read-write mode where it does not release the bus during the cycle. The memory management unit is bypassed so physical addresses can be used without causing segmentation faults. Processor intervention is only required at the end of each block of memory so the load on the CPU is minimal. The processor loading due to the memory wash at 4kbytes/s is less than 2%.

Ramdisk Driver
Reed Solomon block code is used for protecting the OBC386's ramdisk against SEUs. The overhead of this coding scheme is significantly lower (1.5% versus 50%) compared to the more traditional Hamming codes and is practical because the memory will always be accessed in blocks rather than individual bytes.

The scheme works by adding a number of parity words to each block when the block is stored. When the block is read a syndrome is calculated which indicates the presence and location of any errors. Different Reed Solomon codes exist which can detect and correct one or two bits in error per block.

Most Reed Solomon codecs work on blocks of data no bigger than 255 bytes. The QNX file system works on blocks of 512 bytes of data at a time and this is not readily configurable. Therefore, new codes were developed to protect blocks of 512 bytes RS(520, 512) [5]. The code used can detect multiple bit errors and correct up to two errors per block.

In software, the performance overhead was significant and was mainly due to the coding and decoding routines.

An example of a hard disk driver for QNX's file system manager (Fsys) was provided by QSSL so this could be used as the basis for the ramdisk driver. The overheads in accessing the ramdisk through the file system manager and the Reed Solomon coding accumulate to reduce the filesystem throughput (Table 1) from the raw memory bandwidth of 2.6Mbytes/s to approximately 50kbytes/s. Apart from the performance penalty, the coding is transparent to other applications.

The filesystem throughput is important because it ultimately limits the downlink data rate and the speed that images can be transferred from the imaging subsystem. This is not particular to the QNX implementation as the QNX filesystem throughput is similar to the SCOS version (Table 1). A hardware implementation of the algorithm is under development [5] to improve the RS coding time.

	QNX		**SCOS**	
	Write kbytes/s	*Read kbytes/s*	*Write kbytes/s*	*Read kbytes/s*
Raw blocks (fmemcpy)	2600	2600	2600	2600
Filesystem 2kbytes/block RS coding Off	210	207	-	-
Filesystem 2kbytes/block RS coding On	53	52	57	56

Table 1. Filesystem performance for OBC386 at 16MHz

Power Management
Power management is another important requirement for the system. Under SCOS, power control is achieved by halting the processor (executing the HLT instruction) when no tasks are pending. The halt state is detected on the OBC386 and the system clock is reduced to a minimum. However, it appears that it is not possible to execute the HLT instruction from a user task under QNX 4.24 so power control is attained by switching the CPU speed. Table 2 shows the variation on power with processor clock. The power management task slows

the processor clock to the lowest speed (8MHz) when light processor loading is detected. When the processor load increases it increases the processor clock. The power actually consumed depends on the processor loading.

Processor speed	Power [W] No PM	Power [W] PM active
8MHz	3.7	3.7
10MHz	4.1	3.7 - 4.1
16MHz	4.9	3.7 - 4.9
25MHz	5.8	3.7 - 5.8

Table 2. Power consumed at different processor speeds (OBC386 with 64Mb ramdisk, isolated 28V supply)

QNX4.25 and Neutrino now support the Advanced Power Management API and this will be used as the basis for future power management schemes.

Application Software

By providing device drivers to emulate functionality of the existing SCOS operating system the porting effort was significantly reduced. Also, QNX provides communication mechanisms that are similar to those found in SCOS but they are also POSIX compliant. Therefore the applications should be portable to other POSIX operating systems in future.

One of largest, most complex applications under SCOS is a housekeeping task. By allowing a spacecraft operator to access the UNIX shell the complexity of this task has been reduced significantly. Most of the user authentication, system administration and scripting capabilities can now be handled by the shell and the command line utilities.

Future Applications

The 386EX based OBC contains a 10Mbps Ethernet controller and 1Mbps Controller Area Network (CAN) controller. Both networks can be used for inter-OBC communications and control of the spacecraft. The availability of a high speed network means that ramdisks can be shared and processes can be executed on remote nodes to share the computational load. QNX's networking provides fault tolerance [6] as traffic can be automatically switched from Ethernet to the CAN bus in the event of error.

Discussion

Features such as protected mode operation and POSIX compliance were desirable when considering a new operating system for the OBCs. However there were also concerns regarding potential increase in upload time, loss of efficiency and increased power consumption.

The total size of the SCOS operating system image and a basic set of application software is 600kbytes. Under QNX this increases by 25 to 50%. Although this increase is manageable the significance is that at launch the spacecraft is powered down and contains only a bootloader. One of the first tasks after separation is to load the OBC software, including the operating system. If the power to the OBC is cycled during normal operation the software must be reloaded. This is also necessary if the kernel halts due to a software anomaly.

The protected mode operation of QNX makes the last case rare. SCOS has been developed over a long period of time and is very mature and stable but when a new application is introduced there is potential to corrupt the entire system necessitating a complete reload. QNX is still vulnerable to faults in interrupt handlers, remote procedure calls in the filesystem or the downlink server but in most circumstances crashed tasks can be reloaded or restarted.

Performance in some critical areas has been evaluated and shown to be similar to that achieved under SCOS. The Reed Solomon coding is the bottleneck in the filesystem performance and QNX's full POSIX manager does not add significantly to the overhead.

The device drivers, filesystem and network managers use QNX specific code to achieve this performance. Although QNX implements POSIX.1 and part of POSIX.1b these are only a subset of the full operating system [2]. The underlying inter process communication mechanism in QNX is non-POSIX messaging. QNX has the facility for sending messages with no data (proxies) or messages with multiple message buffers. Multi-part messages are used in the QNX I/O managers to achieve high performance [1].

Therefore, there are several choices when implementing custom device drivers. QNX specific functions with a custom interface could be used or a standard I/O driver could be written using POSIX/ANSI C functions only. The choice was made to use QNX specific code for efficiency since some non-portable code was already necessary in order to interface with hardware. The interfaces do not conform to standard I/O drivers because they were written with APIs similar to SCOS. As a result the device drivers and support utilities are portable to other QNX supported platforms with similar peripheral hardware but are not generally portable to other POSIX operating systems. These drivers and utilities took around one working-year of development so represent a considerable investment.

QNX's FLEET™ networking is also not POSIX compliant but provides a powerful mechanism for distributing tasks and sharing resources on multiple OBCs in a fault tolerant manner. The network manager provides a hook for custom protocols so the FLEET networking and other protocols can coexist. A custom file transfer protocol has been developed for interfacing to the Transputer-based image processing system (since the Transputer does not support FLEET). A TCP/IP socket implementation is available which could be used alongside, or in preference to, FLEET to provide greater interoperability between different systems.

Neutrino, QNX's new microkernel, supports multiple platforms. In the next phase of this project Neutrino will be evaluated as a replacement for QNX4.24 and it is expected that this will be a reasonably smooth transition. However, at present Neutrino does not support the processors of interest for the next generation of OBCs (ERC-32 and ARM) so other operating systems will also be considered.

Conclusions

QNX provides a fast, POSIX compatible platform that supports our existing applications. Device drivers were written to emulate some of the SCOS functions to reduce the amount of effort required to port the applications. The micro-kernel architecture has benefits when writing these device drivers as these are essentially user applications. A number of support utilities were required to replace functionality that was built into SCOS.

Reliability is improved in protected mode operation as isolation is provided between the tasks. The use of an operating system that has been proven in many diverse applications provides confidence but also means that it cannot be easily modified. Without access to source code some low level functions could not be implemented but solutions to these problems could be found. However, working with a standard version of the operating system has the advantage that we can readily upgrade to future releases.

Some of the complexity of our own applications could be reduced by using off the shelf utilities that are provided with most UNIX implementations. In addition, new applications exploiting resources on multiple OBCs are now possible using fault tolerant networking.

As the system utilities and operating system are POSIX compliant future changes in processor and operating systems can be accommodated with less effort. The effort is still not insignificant because many of the custom drivers and utilities use operating system and hardware specific functionality.

References

[1] System Architecture, QNX Operating System, QNX Software Systems Limited, 1996.

[2] Frank Kolnick, "The QNX 4 Real Time Operating System", Basis Computer Systems Inc., 1998.

[3] User's Guide, QNX Embedded Kit, QNX Software Systems Limited, 1995.

[4] C.I. Underwood, R. Ecoffet, "Observations of Single-Event Upset and Multiple-Bit Upset in Non-Hardened High-Density SRAMs in the TOPEX/Poseidon Orbit", *IEEE/NSREC Conference*, Snowbird, Utah, USA, Jul, 1993.

[5] M. S, Hodgart and H. A. B. Tiggeler, "Fast Low Complexity Reed Solomon Codec for Space and Avionics Ramdisk Applications", Proceedings DASIA 98, page 95-101, July 1998.

[6] Dan Hildebrand, QNX Software Systems Ltd. "An Architectural Overview of QNX", Proceedings Usenix Workshop on Micro-Kernels & Other Kernel Architecture, Seattle, April 1992.