LINUX POWERED TELEMETRY PROCESSING

Joseph Ayala
Eric Sorton
Command and Control Technologies Corporation
http://www.cctcorp.com

ABSTRACT

Since its debut, the Linux operating system has garnered much attention in the software development community. This paper discusses the open source operating system, Linux, and it's application as the operating system powering a commercial off-the-shelf telemetry processing system.

The paper begins by discussing what are the real-time requirements of the operating system in a telemetry processing system. A discussion to the Linux system is then presented. Soft real-time features of Linux are discussed which allow it to meet the telemetry processing requirements. Linux is compared with the more traditional operating system products and points are made as to why open source software is just as capable, if not preferable, of handling mission critical applications.

The paper also presents the authors' view of future of Linux and open source software in the telemetry marketplace. The paper concludes with a summary of products available for Linux that support telemetry processing and the data acquisition environment.

KEY WORDS

Telemetry Processing, Linux, Open Source, Soft Real-Time.

INTRODUCTION

The goal of this paper is to introduce the reader to the use of the open source operating system, Linux™* and its extensions, as a viable operating system platform for present and future telemetry processing systems. Linux has spearheaded the open source software concept into the limelight of the software development community. Linux is giving legitimacy to the concept of choosing open source software for filling roles in mission critical applications in business, government and the military. While not the first open-source project to be used in mission critical applications, (Sendmail [1] and others have been around longer), Linux is the first to gain significant attention from a more general audience.

* Linux is a registered trademark of Linus Torvald.

Technically, Linux refers only to the core operating system kernel. Hundreds of additional programs and applications, many coded as part of the Free Software Foundations' GNU Project [2] are needed for a functional UNIX* -like system. Common convention is to refer to this collection of utilities, including the core kernel, as Linux. We will use this convention throughout this paper, thus when we are referring to Linux, we are referring to both the operating system kernel and all of the support tools needed to make it a functional system unless otherwise noted.

COTS BASED TELEMETRY PROCESSING SYSTEM

We must begin by first defining what is meant by a COTS based telemetry processing system. In this definition, a telemetry processing system is a collection of Commercial-Off-The-Shelf (COTS) hardware and software which is integrated together to perform the task of acquiring, processing, recording and displaying standard telemetry formats.

A block diagram for a typical COTS based baseband telemetry processing system is pictured in Figure 1. This system will consist of one or more COTS telemetry boards (bit sync, decommutator, and frame buffer combined), a middleware product which is software that processes the telemetry, and some customer specific application programs and displays. The role of the operating system in this system is to provide the glue, which binds the different components of the system together. The drivers, which are integrated into the operating system, bind the telemetry boards to the middleware software. The operating system (OS) provides services to the middleware such as disk I/O, network I/O and inter-process communications (IPC). API's allows the middleware to communicate with user processes such as the applications and/or display programs. It has traditionally been thought that a hard real-time kernel product or UNIX variant OS with hard real-time extensions was necessary for such a system.

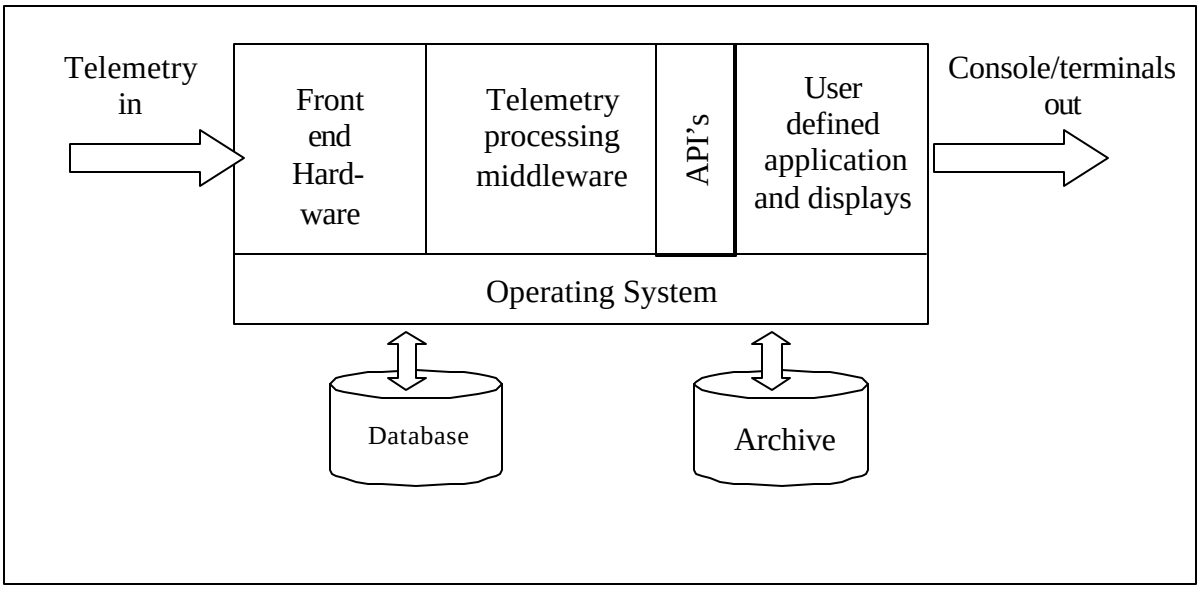

Figure 1. COTS based telemetry processing system

REAL-TIME REQUIREMENTS FOR TELEMETRY PROCESSING

* UNIX is a registered trademark of The Open Group

One question that is paper intends to answer is ”Is Linux real-time enough for telemetry processing?” First let us examine the real-time requirements for telemetry processing.

The amount of time to complete telemetry processing depends on several key factors. One key factor is the bit rate of the telemetry you are processing. Standardized telemetry groups the bits together into minor frames. The minor frame rate (m) is just the telemetry bit rate divided by the number of bits in each minor frame.

$$m = BR / (BW * F) \quad (1)$$

Where: m = minor frame rate in minor frames per second
BR = bit rate in bits per second
BW = bits per word
F = words per minor frame

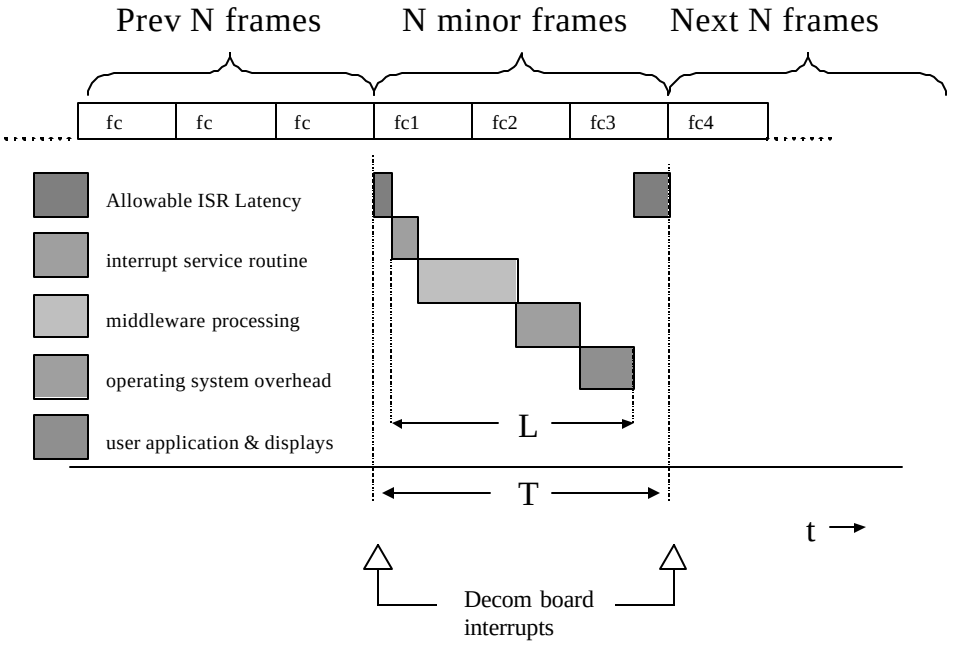

Figure 2 Time allocation for telemetry processing

Figure 2 lays out the time allocation for telemetry processing by the typical COTS based telemetry processing system. Another major factor to consider when determining the time allotted for telemetry processing is the design of the front-end hardware. Most COTS telemetry decommutating hardware will buffer data internally until a set number of minor frames (N) have arrived. A signal or an interrupt, is sent to the OS to remove the unprocessed minor frames from the board while a second or double buffer, continue to receive the next set of minor frames. The decommutator hardware removes the hard real-time constraints from the operating system by managing the strict-timing requirements associated with synchronizing the telemetry stream. By buffering the minor frames to be read, all of the hard real-time constraints are removed from the operating system.

The amount of time available to perform telemetry processing (T) by the middleware and user application is determined by the number of minor frames allowed to be collected in the front end hardware (N),

$$T = N/m \quad (2)$$

Where T = amount of time available in seconds
N = number of minor frames per interrupt from front-end hardware.
m = minor frame rate

The time taken by the system to process the data once it has started processing the interrupt from the front-end hardware is the data latency (L). The telemetry system must continuously perform this processing within the time allotted (T). For a soft real-time system to process telemetry, L must be less than T. If L were too close or equal to T, hard real-time functionality would become necessary. The difference between L and T is the amount of time the soft real-time system time has to begin to execute the interrupt service routine (ISR), which is the allowable ISR latency. If a soft real-time system user can increase the value of N, it would allow the system more time to process each set of minor frames. The downside of making N larger, is of course, longer data latency. Longer latency is not desired in the case where the telemetry system is involved in closed loop digital control. Here there is a desire to minimize (L), since increasing the data latency can result in a loss of control of an external process. For these applications, N is usually is set to 1 and each minor frame is processed as it is received.

What determines the telemetry system's ability to completely process the incoming data within the time T is more a function of processor speed. Thus, in the COTS telemetry processing system described above, soft real-time response are adequate as long as the processing power of the system is sized adequately for the bit rate of the telemetry being received. Obviously, a system based on a 6Mhz 8086 processor will not be able to process and display a 20Mbit telemetry stream.

Now let us examine what facilities are available for Linux to perform soft real-time processing.

REAL-TIME CAPABILITIES OF LINUX

Linux is a UNIX compatible operating system and therefor inherits most of its soft real-time multitasking features from the set of POSIX[3] standards adopted by the IEEE to bring about some uniformity to the diverging set of features being produced by the competing vendors of UNIX operating system. There are a large number of POSIX standards written, but in particular, the standard called POSIX.1b[4] (also known during the development of the standard as POSIX.4) deals with the set of features that bring soft real-time response criteria to the operating system.

Kuhn [5] writes about the availability of POSIX.1b compatibility for Linux. Table 1 highlights at the time of his writing(1998) which of the POSIX.1b features were available written for Linux. They are grouped according to whether they are in the currently distributed kernel, available as patches or libraries that must be added to the distribution, or if it is currently not available.

Available as part of Linux kernel	Available as patches or extensions	Currently unavailable for Linux
Shared memory and memory mapped files (mmap())	Timers (user level)	Asynchronous I/O
Memory Locking(mlock())	Nanosleep()	Prioritized I/O
Priority Scheduling		Shared memory objects.
(Partial) Synchronous I/O (fsync())		Fdatasync()
		Real-time Semaphores, signals
		Message passing

Table 1 POSIX.1b feature availablilty for Linux

Another standard, POSIX .1c deals with system and process threads. While not specified directly as being part of the real-time response, thread implementation goes hand in hand with what is necessary for soft real-time programming. POSIX threads (Pthreads) are now a standard part of the Linux kernel.

Not all of the POSIX.1b features are necessary to implement a soft real-time response. Asynchronous I/O for example, deals primarily with the operating systems ability to optimize buffered disk reads. For telemetry processing, the particular features needed depend on the design of the middleware product. The Command and Control Toolkit (CCTK) is an example of a soft real-time telemetry system running on Linux. CCTK only required the user level timer extension to the standard Linux 2.2 distribution in order for it's middleware product to operate.

None of the POSIX features described above put Linux in the category of a hard real-time operating system (nor is there a standard dictating hard real-time features). To overcome these concerns, research and development is under way to provide real-time operating system extensions to Linux. They are intended to close some of the gaps left by the soft real-time feature of POSIX.1b, in order to create "harder" real-time response for Linux.

KURT and RT-Linux

Hill, Srinivasan, Pather and Niehaus[6] discuss in their paper, KURT, several extensions to Linux that bring a level of real-time response not attainable with POSIX 1.b compliance. They coin the term "firm real-time" to denote a level of real-time response between hard real-time and soft real-time. One extension UTIME, allows high precision timing to occur by running the system timer in a one shot mode rather then continuously. This increases the temporal resolution (timing accuracy) of the system without the overhead that running the timer chip at a faster continuos periodic rate would create. The second extension creates an explicit plan scheduler. This scheduler allows you to create a "plan" which define the real-time tasks and the time required for each. You can then allow any or all processes to be scheduled on a periodic or non-periodic basis. In addition, the developer can choose to use a feature called RTMods, which are loadable kernel modules like device drivers to handle real-time tasks. These modules execute in kernel space and do not require a context switch allowing them to begin execution faster.

RT-Linux[7] uses a different approach to providing real time capabilities to Linux. It uses a small hard real-time kernel which in turn runs the Linux kernel as a low priority task. The primary task of the real-time kernel is handling interrupts from the PC hardware. Communications between the two halves of

the system is through shared memory or queues. Using this approach would require that the real-time application being developed be split into a real-time portion, which requires no Linux services and a non real-time portion that can use the operating system services.

Much of this research into the addition of hard real-time features to Linux is focused on meeting the real-time constraints of embedded machine or process control systems. These are unlike the needs of telemetry processing systems we have been considering. A closer analogy is the real-time needs of multimedia computing. Here data is buffered (usually on disk) and must be processed and delivered to the screen at a sustained continuos rates so that the image or sound do not appear distorted. This can be achieved by insuring there is adequate CPU processing speed to meet the task.

Based upon what has been discussed so far, we can say that Linux, using soft real-time features, is a capable OS for telemetry processing, but it is certainly not the only one. For that matter, almost any operating system conceivable can perform these functions. This is why there are telemetry processing system available on almost every commercial OS. So, real-time capabilities aside, let us focus our attention for a moment to why Linux and open source software provides a better development platform for any software system development, not just telemetry processing.

ADVANTAGES OF USING LINUX

One indisputable advantage of Linux is the source availability. There are several reasons why having the source is a significant advantage. The first, and possibly not so obvious, is that open-source allows software to "evolve". Quoting from Eric S. Raymond's web pages[8], "When programmers on the Internet can read, redistribute, and modify the source for a piece of software, it evolves. People improve it, people adapt it, people fix bugs. And this can happen at a speed that, if one is used to the slow pace of conventional software development, seems astonishing." He continues, "We in the open-source community have learned that this rapid evolutionary process produces better software than the traditional closed model, in which only a very few programmers can see source and everybody else must blindly use an opaque block of bits." Eric S. Raymond does an excellent job of describing the open-source software development model in the paper The Cathedral and the Bazaar.[9] Therefore, even if as a developer or a company you never even sneak a glance at the Linux source code, you still gain enormous benefits from the fact that it is open. Thousands of programmers around the world can continually modify, enhance, and improve open source code, like Linux. Compare that with the number of people assigned to maintain closed source code at any software company.

It will not take long to discover other advantages to having the source code as well. The source code of different aspects of the Linux system make an excellent reference. If documentation is not adequate or if strange behavior is observed in a system and/or library call, it is possible for the developer to reference the actual source to determine what the code is doing. This is not possible with commercial vendors unless an expensive and limiting source code agreement is reached. The source code also becomes an asset when bugs exist. As we all know, software has bugs. At times it is difficult to get some commercial vendor to admit that their product has problems let alone have them fix it. Other times, it may be possible to determine that there is a problem and even get the company to admit there is a problem. However, they may not be willing to fix it. With the source available, it is possible to both find and fix a bug in the source code. In addition, submitting the bug to the maintainer will typically get it into the next minor release of the source. However, you do not need to wait for the next release since you can patch it yourself.

Open source software like Linux encourages improvement and innovation of the existing code. By providing the source code any new innovation or improvement that someone wants to make to a product all effort can be placed into making the improvement. The follow-on developers do not have to re-create the entire product from scratch, which would have to be done if the software was closed source. People developing follow on improvements are free to concentrate on only the improvements, and leave the existing developed and debugged code alone. One may question why someone would want to innovate when there is no profit involved. Thompson[10] answers this question: "the student and moonlighters, motivated by their desire to learn and create and inspired by the energy and clarity of tackling new problems".

One of the common misconceptions about Linux is that it is not supported. Since it is not supported by a single vendor, it is not possible to obtain support. This is untrue. Many commercial entities provide support for Linux. In fact, because the source IS open; anyone with a desire can learn Linux and start a company or consulting business to support it. Additionally, free support is available on-line.

There are other advantages as well, too many to list in this short paper. However, as with anything in life, there are disadvantages as well as advantages and Linux has its share of disadvantages.

DISADVANTAGES OF CHOOSING LINUX

When is it not appropriate to use Linux? Linux is currently not an operating system for the general public. There is a need to twiddle with it enough to require a programmer/engineer on staff for any group that would use it. While many companies are packaging distributions of Linux for more casual computer users, the true advantage of Linux is when there is a environment where people will be developing software. Whether that software is for the telemetry processing or word processing, open source is at its best when there are programmers and administrators familiar to UNIX around.

When a commercial company hires a developer to perform a task, they have control over that individual. They can direct were their efforts are placed to best suit the needs of the company. When a commercial company uses open-source, they do not have that control. They can make suggestions to the open-source developers, but they cannot control them. If the company needs a particular feature for their next release, but the open-source developers wish to code a different feature instead, the company has only two options. Accept that the feature they need will not be completed or do it themselves.

Another disadvantage is deadlines. Whereas the commercial world has deadlines, makes product announcements and promises dates and then, many times, delivers regardless of the condition of the software. The open-source community plods along, release when they finish and has no qualms about delaying a week, a month, or even a year to make sure the product is complete, working, tested and stable. Although this tends to produce extremely stable software, it can can conflict with the desires of a company. Relying on a particular open-source version to be released on a particular date is a formula for failure.

COMPARISON TO OTHER OPERATING SYSTEMS

A detailed comparison of how Linux compares to many of the other commercial operating system is beyond the scope of this paper.[11] What this comparison will attempt is to highlight some of the more popular OS products in the telemetry market and point out similarities or important differences.

VX Works is produced by Wind River Systems.[12] This is a widely adopted "embedded" operating system with full hard real-time support. According to it's marketing literature, VX Works is POSIX 1 and 1b compliant, but it is not a UNIX variant and cannot run programs created for UNIX without extensive modification. In this respect Linux fairs better. Unlike Linux, VX Works runs the kernel and user applications in the same address space. This requires care in user application development since a poorly developed application can cause a system to crash and loose data, violating one of the tenants of telemetry processing. If this feature is desired, the KURT extensions to Linux provide RTMods, which perform a similar function.

QNX by QNX Software System Ltd[13]. Is a small lightweight microkernel architecture that implements the rest is the system functionality with a set of programs called resource managers. The resource managers provide most of the systems POSIX like behavior. This approach is similar to the RT-Linux system. While a formidable following of users have accumulated, QNX still is considered a niche OS and does not have widespread third party support.

UNIX It is hard to compare Linux with all the UNIX variants, because Linux was designed to be a POSIX compliant operating system. All the major features of UNIX are available in Linux. There are even several vendors, LynxOS[14] for example, that provide hard real time extensions to UNIX. There are some defining differences. Many UNIX variants exist, a result of computer hardware manufactures trying to differentiate their system from the others that also run UNIX. This weakens the third party support for UNIX, with vendors producing products for one UNIX variant, but not another. Linux has so far so far been successful at prevented these variants.

Windows NT and Windows 2000 by Microsoft, is a soft real-time operating system, which has, and will continue to have, a significant place in the telemetry market.[15] It's presence of more than 90% of all the computers in the world and the volume of third party software continue to make it a highly desirable platform for many applications. The simple truth is that Windows operating system will always be an option, perhaps even the only option, for almost any product you would need to buy, use or develop.

The emergence of Linux is seen as a way to provide an alternative to users. It first appeared as alternative for user in the operating system market. As Linux's popularity increased, other software vendors began to develop other markets as well. Again the motivation was to provide alternatives to users who wanted more choices. What follows is our opinion on how Linux will continue that trend into the field of telemetry processing systems.

LINUX IN THE TELEMETRY MARKETPLACE

Who will sell Linux based telemetry products? Integrated monolithic telemetry system vendors who provide cradle to grave systems will not port their system to Linux unless they first see demand from the customer base. In all fairness, as long as these vendors continue to improve and provide quality products to their customers, there is no incentive to change. We believe that the first vendors to adopt

Linux will be the software/middleware vendors who see it as an advantage to have their products available on as many platforms as possible. Some third party product developers will choose Linux for the same reason. More will make their software and hardware available when they begin see an established base of Linux system users. By taking advantage of what Linux and open source software has too offer, the developers of telemetry processing software can bring the user community better products. In the end, it will be up to the purchasers those products to decide whether to change or remain with the status quo. Therefor, it is possible that the marketing of Linux based products may require overcoming some of the fear, uncertainty and doubt (FUD) that is being spread about open source software. Throughout the rest of the software industry, entrenched interests intent on keeping the status quo try to dissuade users from changing by painting Linux as incapable of fulfilling mission critical roles. While there is no evidence that this will happen, as with other areas of software, it may likely be the case with telemetry systems as well.

In reality, open source software is already serving mission critical roles in other industries. More than half the Internet Web servers are based on the open source server, Apache. The open source program sendmail, handles most of the world's e-mail. FUD must be avoided because it does not serve the interest of the user. It is possible that if the user community were to request Linux based telemetry products, less energy will be spent of FUD, and more effort would be placed on bringing the benefits of Linux and open source software to the telemetry processing community.

A brief market survey was conducted in April 2000 to gauge the availability of "Linux powered" products in the telemetry marketplace. This survey is by no means exhaustive. It was done using Internet Web search engines using search words "telemetry" and "Linux", and by browsing telemetry system vendors Web sites.

Command & Control Toolkit CCTK for Linux CCT has successfully ported the Command and Control Toolkit to both Intel and Alpha Linux. In both cases, the port was relatively simple with no significant problems related to the operating systems themselves. Two customers are taking advantage of Linux for use in their systems. First, CCT is building a telemetry simulator using CCTK and Intel Linux for a NASA customer. Second, CCT is building a launch control system for Spaceport Florida Authority's Complex 20. This system will run CCTK on an Alpha Linux system.

Satellite Toolkit Analytical Graphics has announced a version of their flagship product Satellite Toolkit (STK) for Linux. STK is used to calculate and monitor orbital dynamics of space vehicles.

RACSI[16] The European Space Agency (ESA) has chosen to deploy a Linux based laptop which will operate inside the international space station (ISS) by an astronaut to allow the surveillance and control of the rendezvous and docking operations of the ATV, ESA's automatic transfer vehicle.

CONCLUSION

This paper has shown that Linux is capable of handling the demands of telemetry processing for a COTS based telemetry processing system. We have shown that soft real-time response, built into Linux is all that is necessary to meet the requirements of telemetry processing. We have also shown that hard real-time extensions exist if they are needed. As a software development platform, Linux has distinct advantages over other closed source operating system that can be used by telemetry processing software developers to to bring highly reliable, quality products to the telemetry market.

REFERENCES

[1] The Sendmail Homepage. Homepage on-line at http://www.sendmail.org/.

[2] GNU's not UNIX! – the GNU Project and Free Software Foundation (FSF). Homepage on-line at http://www.gnu.org/.

[3] http://stdsbbs.ieee.org/

[4] B. Gallmester. POSIX.4: Programming for the Real World O'Reilly and Associates, 1995

[5] Markus Kuhn. A Vision for Linux 2.2 – POSIX .1b Compatibility and Real-Time Support. Online at ftp://ftp.informatik.uni-erlangen.de/local/cip/mskuhn/misc/linux-posix.1b

[6] Robert Hill, Balaji Srinivasan, Shyam Pather, and Douglas Niehaus. Temporal Resolution and Real-Time Extension to Linux. Technical Report ITTC-FY98-TR-11510 Dept of Electrical Engineering and Computer Sciences, University of Kansas. June 3, 1998

[7] V. Yodaiken. The RT-Linux approach to hard real-time. Available online at http://www.rtlinux.org/rtlinux.new/documents/papers/whitepaper.html

[8] Introduction to Open-Source. On-line at http://www.opensource.org/intro.html.

[9] The Cathedral and the Bazaar. Eric S. Raymond. On-line at http://www.tuxedo.org/~esr/writings/cathedral-bazaar/cathedral-bazaar.html.

[10] Bernie Thompson. Making Money in the Bazaar, Part 1. *Linux Journal*. June 1999, Issue 62.

[11] http://www.realtime-info.com/encyc/market/rtos/rtos.htm

[12] http://www.windriver.com/products/html/vxwks54_ds.html

[13] http://www.qnx.com/literature/whitepapers/archoverview.html

[14] http://www.lynuxworks.com/

[15] M. Wexler. The Future of Data Acquisition Proceedings of ITC 98 Volume XXXIV

[16] http://www.estec.esa.nl/wawww/ESC/racsi.html