Discussion of Microkernel and Monolithic Kernel Approaches

Ivan Stankov, Grisha Spasov
Technical University – Sofia, branch Plovdiv, Faculty of Electronics and Automation, Plovdiv, Bulgaria

***Abstract**: The paper discusses monolithic approach against microkernel approach - both used in operating systems. The paper presents the most critical features of operating systems concerning system architecture, communication model, and priority in processes, memory management and handling of any arrived exceptions as well as interrupts. The interrupt is a crucial factor for real time operating systems (RTOS). It is supposed the interrupts to be handled in real time for predictable time. The paper points out the benefits of each discussed approach and its weak points. At the end a conclusion for each appropriate operating system deployment is made, based on facts and results, and also on prediction for functionality. The paper discusses also the embedded systems as one of the most fast developing systems and mostly suitable platform for any RTOS installation or any other operating systems (OS) functionality.*

***Keywords:** real-time operating systems, monolithic kernel, microkernel, architectures, organizing distributed networks*

1.INTRODUCTION

Nowadays more and more resources are thrown for improving performance of existing software/hardware systems to produce new ones more powerful than ever. In the world of embedded systems and real-time systems anything is crucial – software applications, involving operating systems as well as hardware platforms. In some cases the terms of embedded systems and real time systems are overlapped, but mostly it is not. The most significant feature that distinguishes real-time systems from non real-time systems is any predictability. A real-time system responds in a (timely) predictable way to all individual unexpected external stimuli arrivals. In that case the average performance is not an issue.

In operating systems issue for embedded systems platforms are existing QNX Neutrino – RTOS, Linux based distributions, implementing basic functionality as well as Embedded Windows or Windows CE. The last two OSs are not discussed in current paper, because they require high system resources to the deploying platform. The purposes of current operating systems discussion is for comparing and pointing out the main features of aforementioned OSs for any appropriate deployment onto an embedded platform for achieving the best performance on reasonable price.

2.BACKGROUND

In any cases software applications that works onto an appropriate hardware platform is characterized with some specific features. These features are deployed into these applications that work on. But with increasing computing power of embedded systems some standards have to be involved and some development environment is needed. In that reason middleware [4] as a technology and a standard become very significant in N-tiers [1] model and heterogeneous systems. It is the semantic glue [1] for different platforms and standards and that all based on XML technology [1, 10] for common cases. Because of these reasons a couples of initiatives were settled and

projects were undertaken by universities and companies with the responsibility to provide standardization for developing of embedded systems applications.

A group of European companies and research institution, part funded by European Commission, has launched an initiative, and dubbed SPEED – Speculative and Exploratory Design in Systems Engineering, which aims to define a standard methodology for the creation embedded systems design. It is a concerted effort to define a standard, end-to-end framework for the implementation of the next generation concept, methodologies, processes, technologies and tools for the design of embedded systems. The initiative's aim is also to provide an environment that will foster greater collaboration between European companies of all size involved in embedded systems.

RUNES project [5] has a vision to enable the creation of large-scale, widely distributed, heterogeneous networked embedded systems that interoperate and adapt to their environments [1, 7]. The inherent complexity of such systems simplifies programmers to use all available potential for realizing of networked embedded systems. The widespread use of network embedded systems requires a standardized architecture which allows self-organization to suit a changeable environment what RUNES have learned from existing middleware solutions. Their aim is to provide an adaptive middleware platform, a common language that will simplify the application creation process.

Two approaches for realization of any standardization exist: open source, which is freely developed by an appropriate initiative or volunteers, and firmly strongly organized and supported, which costs a great value. However, on the other hand the problems met with open source initiative are inevitable and unpredictable, while the problems with firmly supported software are expected and well documented.

3.OSS' ARCHITECTURE APPROACHES AND FEATURES

There are many ways to add real-time capabilities to Linux/UNIX or other OS based systems. Certainly, one method is to throw hardware at the problem, by running faster processors or employing specialized hardware. For example, specialized peripheral controllers and digital signal processors (DSPs) can offload critical real-time tasks from the main system CPU. However, assuming you want to use the main system processor to manage real-time system events, there are a great many options from which to choose.

This paper presents comparison of two architectures for building a real-time operating system microkernel and monolithic kernel. As a presenter of the first one QNX Neutrino is discussed and performed results involve v 6.2. [8] For the second one Red Hat Embedded Linux (ELDS 1.1) [9] is discussed.

3.1.Microkernel

QNX Neutrino is based on real client/server architecture and consists of microkernel and optional cooperating processes. The microkernel implements only the core services, like threads, signals, message passing, synchronization, scheduling and timer services. Additional functionality is implemented in cooperative processes, which act as server processes and respond to the request of client processes. In this case the application or any other functional modules act as a client. This technique is based on message oriented communication model (Fig.1). Communication model uses message bridges for message transfer to any node in the network. In that way the distributed character of QNX is deployed in its design. The distributed nature defines the distribution of tasks among all machines and that nature allows all resources to be

distributed in the network. In that way only one machine/system has to owe appropriate feature. It is possible that feature to be shared over the network to other machines. For instant only on one system needs to be set network adapter, in that way all others will use it as gateway for communication outside the local network.

One very important feature of the QNX architecture is priority organization. While the kernel runs at privilege level 0 of the Intel processor, the managers and device drivers run at levels 1 and 2 (to perform I/O operations). Application processes on the other hand run at privilege level 3, and can therefore only execute general instructions of the processor.

Another feature in architecture of QNX is the presence of own virtual space to each process that is started on that platform. Involving the features of communication model the accomplished system is very scalable and flexible with distributed nature that is crucial for embedded systems.

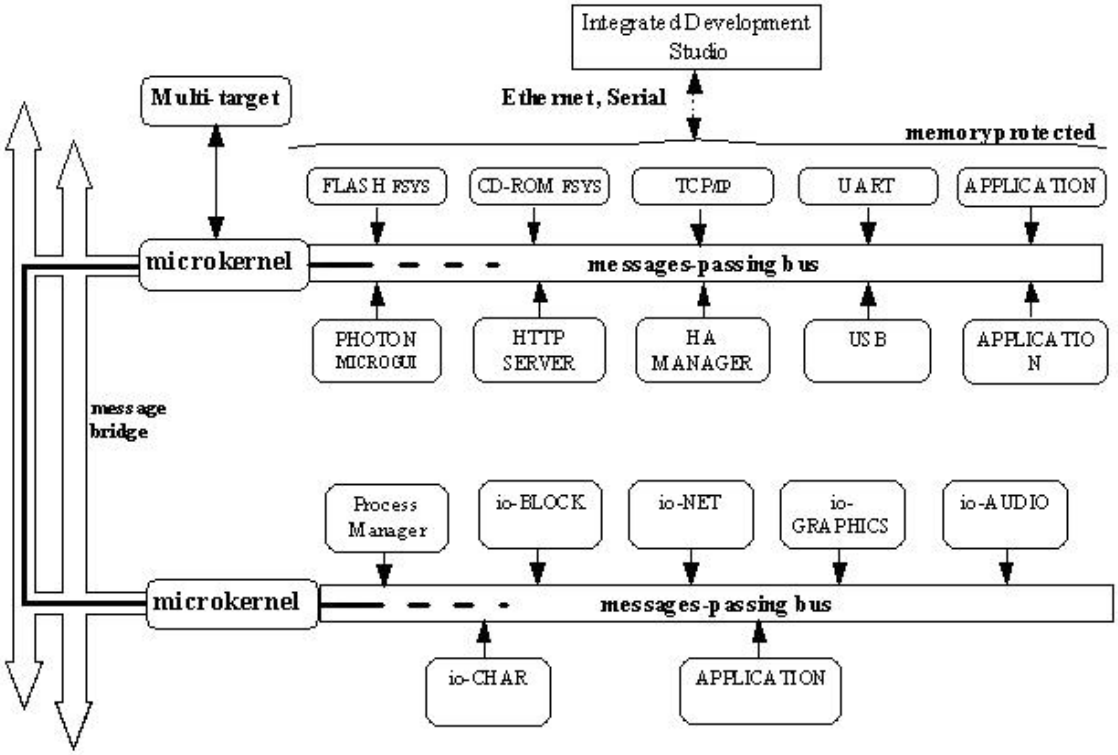

Figure 1. Microkernel architecture presents distributed design based on communication model that uses Message Bridge for message transferring

QNX Neutrino can reach as possible response time as can perform the hardware on which the discussed RTOS is working on. That could be achieved via fast context switching. Distributed priority inheritance is performed by QNX. This means that each instance of which any resource is called has the same priority. Combining that feature with message driven communication model, a priority orientation in fast contest switching is achieved, that is crucial for embedded systems where the time diagrams are very precisely driven.

The QNX Neutrino is all POSIX standards compatibility available and supported. The discussed architecture allows distributed programming facilities without any appropriate skills and knowledges. This is essential for embedded systems that posses limited resources. In that way QNX is very appropriate for systems with limited resources because of the possibility to load in involve as much functionality as it is needed and the size of OS is as much as possible for the deploying platform.

3.2.Monolithic kernel

A monolithic kernel is a kernel architecture where the entire kernel is run in kernel space in supervisor mode. In common with other architectures (microkernel, hybrid kernels), the kernel defines a high-level virtual interface over computer hardware, with a set of primitives or system calls to implement operating system services such as process management, concurrency, and memory management in one or more modules.

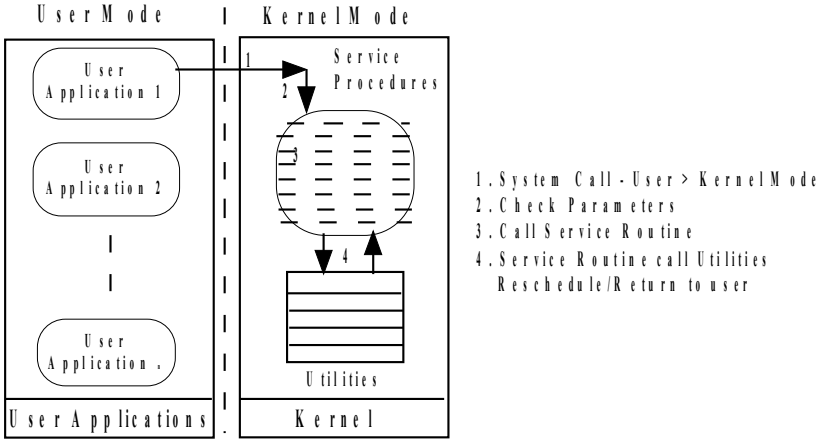

Figure 2. Monolithic Architecture

Even if every module servicing these operations is separate from the whole, the code integration is very tight and difficult to do correctly, and, since all the modules run in the same address space, a bug in one module can bring down the whole system. More modern monolithic kernels such as Linux, FreeBSD and Solaris can dynamically load (and unload) executable modules at runtime. This modularity of the kernel is at the binary (image) level and is not at the kernel architecture level. The two can be completely independent, but are sometimes confused. Modular monolithic kernels are not to be confused with the architectural level of modularity inherent in microkernel or hybrid kernels.

ELDS v1.1 uses the Red Hat Linux 7.2 kernel which is based on the general Linux kernel v2.4.5. Linux has its roots in the Unix General Purpose Operating System (GPOS). It has a traditional monolithic kernel and is clearly not built for real-time purposes. Fortunately, Red Hat does not claim real-time behavior for the Linux kernel used by the ELDS. But on the other hand Linux is process based and has virtual memory protection between the different user processes and between a user and kernel process. The kernel can be extended by so called "modules" (for instant device drivers). The kernel is well protected against faults in such modules. Swapping can be avoided by not mounting a swap file-system. However the Linux based kernel is not built for real-time purposes and because of that the kernel is not pre-emptive and is still a monolithic kernel, using large and complex shared data structures resulting in critical sections, which are large and can lock the processor for a considerable amount of time. Moreover this OS supports primitive kernel threading and processes that affect on other APIs that are built on calls of these primitives. Another effect of not real-time nature of the Linux based kernel is no priority inversion avoidance mechanism. This is a problem in all GPOSs. As these OSs are not built for real-time systems, there is no need for the rather complex protection against priority inversion.

4.INTEGRATED DEVELOPMENT SUITES (IDES)

IDE is very significant attribute to any operating system. It helps a programmer to develop applications easily and straight forward. Debugging is a very important process that sometimes takes a long period of time. QNX operating system has its own Development Suite for application developing, as well as Red Hat Embedded Linux Developer Suite version 1.1 and Windows' Visual Studios. Each of these studios makes the process of developing and deploying very easy.

QNX Development Suite is possible to be installed on Windows, Linux, Solaris and MacOS platforms. Applications could be developed even on local machine, while Red Hat Embedded Linux is host development driven and uses GNU tools, graphical front-ends that have been developed Source Navigator or KDevelop C/C++ IDE, DDD or Insight front-end for gdb. Most of these tools are integrated in the Red Hat distribution and the developer has plenty of choice. Installation of the tools is easy thanks to the rpm package management system.

5.COMPARISON RESULTS, CONCLUSIONS AND FUTURE WORK

On base of our experience in distributed embedded systems and N-tiers models some conclusions are made concerning mostly these architectures. We tried to make comparison on the most significant features of operating systems like: installation and configuration, RTOS architecture, internet support, available tools, documentation and support. The results presented in table 1 and shown on figure 3 are taken from [8, 9].

Table 1 Comparison of significant OS's features

OSs Features	QNX 6.2	ELDS v1.1
Maximum Sustain Interrupt Frequency	9 µs	60 µs
Clock	100 %	43%
Memory management	27 %	0 %
Interrupt handling	88 %	63 %
Semaphores	35 %	31 %
POSIX Signals	100 %	78 %
Mutex	67 %	31 %

Linux based systems have no real-time behaviour and they are license fee free, but the embedded toolset is not. Its weak points are providing of little or no assistance to configure the embedded target's kernel; there is no sufficient documentation provided with the OS; API is not compliant with POSIX standards. QNX is firmly supported and has all real-time advantages including fast performance, excellent architecture for a distributed system and distributing in general and robust system and Good platform support. Based on comparison between two approaches a conclusion could be made that the both systems show very strong performances, but when it is needed a real-time approach QNX is the more sensible and reasonable choice. On the other hand, if no real-time requirements are considered necessary then open source solution is better alternative, mostly because no license agreement is needed.

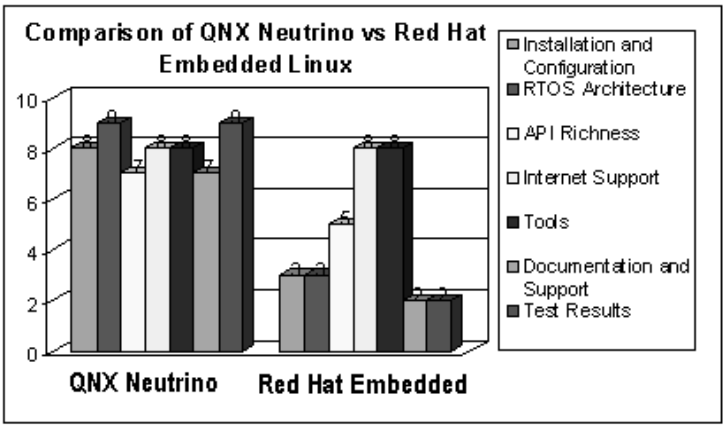

Figure. 3. Comparison between QNX Neutrino 6.2. and Red Hat Embedded Linux

In table 1 are presented some of really significant feature of discussed approaches presented by QNX 6.2 and ELDS v1.1 the data is taken from [8, 9]. It presents the faster and more reliable approach of microkernel – interrupt frequency and interrupt processing. Better memory management, which is crucial for limited resources of embedded systems. POSIX compatibility of API functions of microkernel approach give to programmers free to access critical platform resources.

6.ACKNOWLEDGEMENT

The presented work is supported by National Science Fund of Bulgaria project "ВУ-966/2005", entitled "Web Services and Data Integration in Distributed Automation and Information Systems in Internet Environment", under the contract "ВУ-МИ-108/2005".

7.REFERENCES

[1] Kakanakov, N., M. Shopov, G. Spasov (2006), *A New Web-based Multi-tier Model for Distributed Automation Systems*, Information Technology and Control (J.), (in press).

[2] Kakanakov N., I. Stankov, M. Shopov, and G. Spasov, (2006) *Controller Network Data Extracting Protocol – design and implementation*, Proceeding of CompSysTech'06 , V. Tarnovo, pp.III-A.14-1-6.

[3] Labrosse J.J., (2002) *MicroC/OS-II - The Real Time Kernel*, CMP Books, USA, ISBN:1-57820-103-9

[4] Lea, D., and S. Vinoski, (2003) *Middleware for web services, IEEE Internet Computing*, vol 7, no 1, pp 28-29

[5] Reconfigurable Ubiquitous Networked Embedded Systems (http://www.ist-runes.org/)

[6] Stallings, W. (2002), *High-Speed Networks and Internets*, 2nd Ed, Prentice Hall, ISBN: 0-13-032221-0.

[7] Stankov I. and G. Spasov, (2006) *Web based application for distributed remote measurement viewing*, Proceedings of ICEST, Sofia, pp.336-339.

[8] The QNX NEUTRINO RTOS v6.2 evaluation report, Dedicated Systems Experts, 2002, (http://www.dedicatedsystems.com)

[9] The Red Hat ELDS v1.1 evaluation report, Dedicated Systems Experts, 2002. (http://www.dedicatedsystems.com)

[10]XML Standard: http://www.xml.com/