Design and architecture of real-time operating system

[1, 2] *K.M. Mallachiev <mallachiev@ispras.ru>*
[1, 2, 3] *N.V. Pakulin <npak@ispras.ru>*
[1, 2, 3, 4] *A.V. Khoroshilov <khoroshilov@ispras.ru>*
[1] *Institute for System Programming of the RAS,*
25, Alexander Solzhenitsyn Str., Moscow, 109004, Russia.
[2] *Lomonosov Moscow State University,*
GSP-1, Leninskie Gory, Moscow, 119991, Russia.
[3] *Moscow Institute of Physics and Technology (State University)*
9 Institutskiy per., Dolgoprudny, Moscow Region, 141700, Russia
[4] *National Research University Higher School of Economics (HSE)*
11 Myasnitskaya Ulitsa, Moscow, 101000, Russia

Abstract. Modern airliners such as Airbus A320, Boeing 787, and Russian MS-21 use so called Integrated Modular Avionics (IMA) architecture for airborne systems. This architecture is based on interconnection of devices and on-board computers by means of uniform real-time network. It allows significant reduction of cable usage, thus leading to reducing of takeoff weight of and airplane. IMA separates functions of collecting information (sensors), action (actuators), and avionics logic implemented by applied avionics software in on-board computers. International standard ARINC 653 defines constraints on the underlying real-time operation system and programming interfaces between operating system and associated applications. The standard regulates space and time partitioning of applied IMA-related tasks. Most existing operating systems with ARINC 653 support are commercial and proprietary software. In this paper, we present JetOS, an open source real-time operating system with complete support of ARINC 653 part 1 rev 3. JetOS originates from the open source project POK, created by French researchers. At that time POK was the only one open source OS with at least partial support for ARINC 653. Despite this, POK was not feasible for practical usage: POK failed to meet a number of fundamental requirements and was executable in emulator only. During JetOS development POK code was significantly redesigned. The paper discusses disadvantages of POK and shows how we solved those problems and what changes we have made in POK kernel and individual subsystems. In particular we fully rewrote real-time scheduler, network stack and memory management. Also we have added some new features to the OS. One of the most important features is system partitions. System partition is a specialized application with extended capabilities, such as access to hardware (network card, PCI controller etc.) Introduction of system partitions allowed us moving large subsystems out of the kernel and limiting the kernel to the minimal functionality: context switching, scheduling and message pass. In particular, we have moved network subsystem to system partition. This

moving reduces kernel size and potentially reduces probability on having bug in kernel and simplifies verification process.

Keywords: ARINC 653; RTOS; IMA; partitioning; real-time.

1. Introduction

Real-time Safety-critical systems have strong requirements in terms of time and resource consumption. Most of them have several concurrently executing separate functions (applications), which communicate from time to time. The most obvious approach is running those applications on separate devices and connecting to sensors and actuators by point-to-point link, on which applications should communicate. But firstly, there will be a lot of wires in large system. And secondly, having a separate computing node for periodic application, which is idle most of the time, results in a great number of computing nodes and high cost of hardware.

Integrated modular avionics (IMA) network is a solution to those problems in avionics. Core modules are main part of IMA network. Core module runs a real-time operating system (RTOS), which supports independent execution of several avionics applications that might be supplied by different vendors. System provides partitioning, i.e., space and time separation of applications for fault tolerance (fault of one application doesn't affect others), reliability and deterministic behavior. The unit of partitioning is called *partition*. Basically partition is the same as process in commodity operating systems. ARINC 653 standardizes constraints to the underlying RTOS and associated API. [1]

Civil aircraft airborne computers are mostly PowerPC architecture. In this paper we present the project on development of an open source ARINC 653 compatible operating system, which can run on PowerPC CPU and, in the future, on other CPU architectures, such as MIPS and x86.

1.1 Overview of ARINC 653

ARINC 653 is the standard for implementing IMA architecture; it defines general purpose APplication Executive (APEX) interface between avionics software and underlying real-time operating system, including interfaces to control the scheduling, communication, concurrency execution and status information of its internal processing elements.

Key concept of ARINC 653 is partitioning of applications in integrated module by space and time. [2]. A partition is a partitioning program unit representing an application. Every partition has its own memory space, so one partition cannot get access to the memory of another. Partitions are executed in user (non-privileged)

mode, so errors in partition cannot affect OS kernel (which is executed in privileged mode) and other partitions. Partition consists of one or more processes, which operate concurrently. Processes in partition have the same address space and can have a different priority. Process has an execution context (processor registers and data and stack areas), and they resemble well-known concept of threads. Fig. 1 shows example architecture.
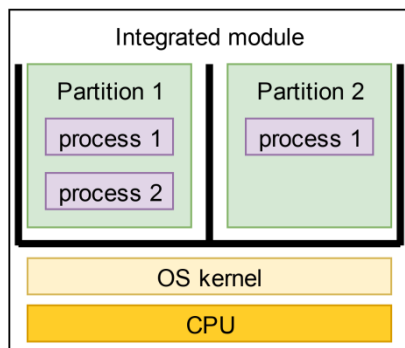

Fig. 1. Example module architecture

Partitions are scheduled using a simple round-robin algorithm. System defines a major time frame of fixed duration which is constantly repeated through integrated module execution time. Major frame is divided into several time windows. Each partition is assigned to one or more time windows, and partitions are running only during corresponding assigned time window. Assignment of time windows and major frame duration are statically configured by the system integrator, therefore scheduling is fully deterministic.

Scheduling of processes within partition is a dynamic priority based scheduling and communication and synchronization mechanism make it more sophisticated than partitions scheduling.

ARINC 653 provides interface for communication between applications (partitions), potentially running on different modules connected by onboard communication network. All inter-partition communication is conducted via messages. Message is a continuous block of data. The ARINC 653 interface doesn't support fragmented messages. Message source and destination are linked by channels; a channel links a single source to one or more destinations. Partitions have access to channels via defined access points called ports. Port has single direction; it can be either source or destination port. One port can be assigned only to one partition. Each partition can have multiple ports. It is even possible to have a channel where both source and destination ports are assigned to one partition

Partition code works with ports regardless of underlying channels. Channels are preconfigured statically.

To control the concurrent execution of processes ARINC 653 offers synchronization primitives such as semaphores, events and mutexes. Buffers and blackboards provide inter-process communication within a partition. Buffer is a messages queue, while blackboard has only one message, which is rewritten by every write operation.

2. Related works

ARINC-653 requirements results in constrains to underlying operating system. OS must support:

- space partitioning, so partitions have no access to memory areas of the other partitions and OS kernel;
- time partitioning, so not more than one partition can run at any time;
- strict and determinate inter-partition scheduler that ensures application response time.

Furthermore, in safety-critical systems the operating system must undergo certification process. As a result, size and complexity of OS become a real issue.

Popular real-time operating systems (such as RTERMS [3] and FreeRTOS[4]) don't support ARINC 653. Furthermore, RTERMS doesn't support memory protection.

Operating systems that satisfy all of these constrains are exist, but they are commercial and proprietary software. They are VxWorks[5] (by Wind River), PikeOS[6] (by Sysgo), LynxOS [7](by LynuxWorks).

There are research projects on real-time and ARINC 653 [12] enhancements of Linux. But Linux is a large system, so certification of Linux kernel seems impossible.

There are research projects that exploit the virtualization technology to support ARINC 653. But they are either proprietary like LithOS[8] (works over open hypervisor XtratuM[9]), or limited prototype VanderLeest implementation of ARINC 653 over Xen [10].

Only POK operating system [11], which is available under BSD license terms, mostly satisfies our requirements, so we decided to fork POK and continue its development.

3. POK

POK is a partitioned operating system focused on safety and security [11]. We describe it in detail here since it is the basis for the JetOS that we are working on.

POK has been designed for x86 and ported to PowerPC (PReP) and Sparc. POK has two layers: kernel and partition, where services of partition layer run at low-privileged level (user mode), and kernel services are executed at high-privileged level (kernel mode). Besides the kernel POK provides a library for partition code (libpok), which translates ARINC 653 API to POK kernel syscalls. Fig 2 shows POK architecture.
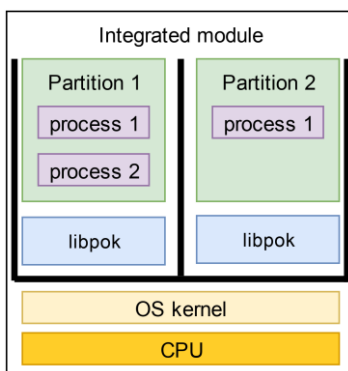
Fig. 2. POK architecture

We selected POK as the basis for our RTOS. Below in this paper we describe parts of POK that were changed or rewritten. We describe limitations of current implementation or architecture of these parts.

Partition management. POK provides partition isolation:

- in time by allocating fixed time slots for partitions in the schedule,
- in space by associating a unique memory segment to each partition.

Partition scheduling and memory management of POK partly comply the ARINC 653 specification. But PowerPC processor, on which we focus (P3041), doesn't support memory segmentation.

Processes management. POK supports ARINC 653 partition processes. All processes are represented in the kernel as array entries of a single processes array that stores process information for all partitions. POK has no logical separation in kernel representation of ARINC-653 processes of different partitions.

POK supports two intra-partition schedulers: Rate Monotonic Scheduling (RMS) and Earliest Deadline First (EDF). Those partitions schedule processes within a partition when its time slot is active.

The problem with POK scheduler is that ARINC 653 requires much more from intra-partition scheduler: priority scheduling and fault management.

POK runs both inter- and intra-partition schedulers in the kernel mode.

Inter-partition communication. For every ARINC port there is a buffer of corresponding size inside the kernel. User code while sending to (or receiving from) port accesses those buffers by means of syscalls. At the beginning of every major frame POK copies data from source buffers to destinations. For large buffers there is possibility to spend significant part of partition time slot on buffer-to-buffer copying.

If a process tries to send to a full port (or read from an empty one) the kernel blocks the process until buffer becomes operational. POK supported this feature but did not

obey to the ARINC-653 requirement on that the order of unblocking should be the same as the order of blocking on each priority level.

Intra-partition communication support is implemented by the user-mode library libpok, using system calls for synchronization purpose. It supports locking resources for concurrent access to shared data resources (such as buffer and blackboards) between processes in partition. When process tries to accesses a locked resource, it will be blocked (so scheduler will skip this process) until the resource is unlocked.

POK scheduler has some inherent problems with handling of locked processes. Let's consider an example. A low-priority locks a buffer for writing and before it unlocks the buffer a higher priority process wakes up. POK scheduler unconditionally switches to the second process. If the second process tries to get status information about the locked buffer it blocks and POK wakes the first process. But according to ARINC-653 standard the process that requests status information must not block.

4. JetOS

JetOS is the real time operating system with ARINC-653 support that we currently develop at ISPRAS. It originates from POK but has evolved significantly since then.

Before we introduce the new features of JetOS compared to POK let us mention the facility that was removed from POK: the AADL configuration tool. Originally POK was designed and implemented as a demonstration of a number of approaches, and the developed selected rather exotic approach to configuration. The suggested way to create an embedded application by means of POK is to specify its environment and capabilities as an AADL specification. In JetOS we dropped AADL support in favor of XML-based configuration files.
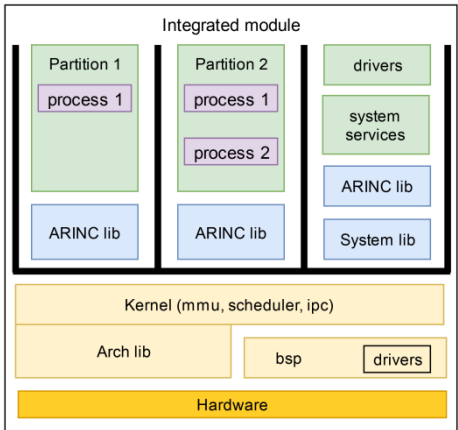

Fig. 3. JetOS architecture

Furthermore, we dropped support of the SPARC platform as there are no onboard avionics systems that are built atop of SPARC CPUs. At the moment JetOS runs on x86 and PowerPC (Book E branch).

Partition management. Unlike x86 and SPARC the new target hardware for JetOS, PowerPC platform, features direct MMU control through TLB writes. To reduce cache flushes at context switches and simplify TLB lookups PowerPC provides tagged cache where each tag is an 8 bit identifier. We use that identifier as partition identifier (pid). At context switch we just change value of the special-purpose register responsible for current pid. This is simple and secure method.

The inter-partition scheduler of POK was able to switch partitions only when the active process runs in user mode. If a process calls syscall it cannot be switched until the end of that call. Such behavior violates requirements of real-time since system calls might be prolonged. Currently we are working on kernel-mode critical section and synchronization primitives to enable context switch while a process executes a system call.

Processes management. We store process-related data in kernel separately for different partitions. Intra-partition scheduler was fully rewritten to support ARINC 653 specification. The new scheduling facility allows for multiple schedulers, and different partitions might utilize different schedulers (a.g. ARINC-653 for avionics applications and preemptive pthreads for system partitions). New intra-partition scheduler can be accessed only by functions

- start() is called when partition is starting or restarting
- on_event() is called on every event such as timer interrupt and returning control to partition.

Inter-partition communication. We use one ring buffer for every channel. Its size is the sum of source and destination ports buffers size in original POK design. It removes the need for copying from source to destinations buffers. Correct work of send and receive function achieved by two pointers, one for source port, and one for destination. Sending increases source port pointer, receiving increases destination port pointer. When pointers are met then buffer either full or empty, uncertainty is resolved by another variable associated with the channel, which stores current number of messages in the channel's buffer. Example can be seen at Fig. 4.
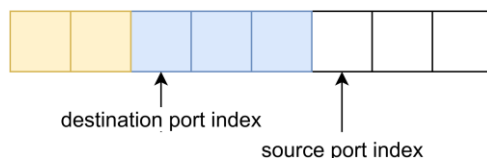

Fig. 4. Example kernel channel buffer. Yellow cells are already received messages, blue cells are sent but not yet received messages, white cells are empty

Intra-partition communication. Correct handling of concurrent data access to buffers and blackboards without violating the ARINC 653 scheduling requirements

with user mode scheduler is a hard task. Therefore the intra-partition schedulers are implemented in the kernel to simplify lock-wait-unlock and priority scheduling. In future versions we may design a solution that solves this issue while keeping a code in user space.

4.1 Configuration

The characteristic feature of real-time operating systems is deterministic behavior. The primary way to ensure reliable and dependable behavior is static pre-allocation of all resources – memory, CPU time, access to devices, etc. For instance, partition code is executed only during fixed time slots within the schedule, no sooner, but no later. Memory is pre-allocated for every partition, memory image of the partition is fixed, no pages could be added or removed during runtime.

Many parameters of our operating system are configured statically and cannot be changed dynamically. These parameters are number of partitions and their memory size, number of ports, their names, sizes and directions, channels etc.

Configuration of the system is stored in xml documents. To keep the kernel minimal we got rid of the need to include xml parser to kernel: the configuration files are processed at build time. The processor generates C code where parameters are presented as either preprocessor macros (#define constants) or enum constants. The generated files are included in the build process.

4.2 System partitions

Beside ordinary partitions, that interact with the kernel and the outer world thought ARINC 653 APEX, the standards allows for so called *system partitions* that utilize interfaces outside the scope of APEX services, such access to devices or network sockets. The standard doesn't specify their operations and interfaces other than constraints on time and space partitioning: system partitions are subject to scheduling. The difference between system partitions and kernel modules is that system partitions run in user space and have time and space partitioning constrains.

Our OS supports system partitions. From the kernel point of view system partitions are like ordinary partitions with some additional memory mapping and additional system calls. Communication between application partitions and system partitions is performed through ARINC-653 ports.

Currently we have only one system partition: the IO partition that is responsible for communication over the network. In the future we will implement a number of other system partitions – file system, graphics server,

IO partition has access (by corresponding entry in TLB) to special memory areas, where network card registers are mapped, so IO partition can work directly with hardware without kernel system calls.

IO partition receive and send data either from partitions in the same integrated module by ports or from other integrated modules by network card drivers. In the simple case the communication over network is based on UDP messages, and the configuration

defines mapping between ARINC 653 port and a pair of IP address and UDP port. This mapping looks like ARINC channel, so we also call it channel.

But network communication may be based on other protocols, such as AFDX. So in general, the channel maps ARINC port to some network specific data. We support parallel work with several network protocols, by assigning channel driver to channel. Channel driver is interlayer between port and device driver. In most cases channel driver is a network stack.

System can have several network cards, so we support parallel independent work of several device drivers. Currently we support three network cards drivers: virtio, ne2k family and hardware cards on the platform with P3041 processor.

Each network driver manages one or more uniform devices. During initialization each driver, which cards are connected through PCI bus, registers as PCI device in PCI driver. After initialization of all network drivers PCI driver starts enumeration of PCI bus. If it finds a physical device that matches a registered PCI device, then it signals to the corresponding network driver. Network driver dynamically for every signal registers a *network device*. Network device has a name and method to send and receive data from assigned physical device. Names to network device are assigned dynamically; name is concatenation of drivers name and sequential number of current device in driver.

The configuration assigns channel drivers to network devices by name. Example of sending two messages in parallel to two different network cards can be seen at Fig. 5.
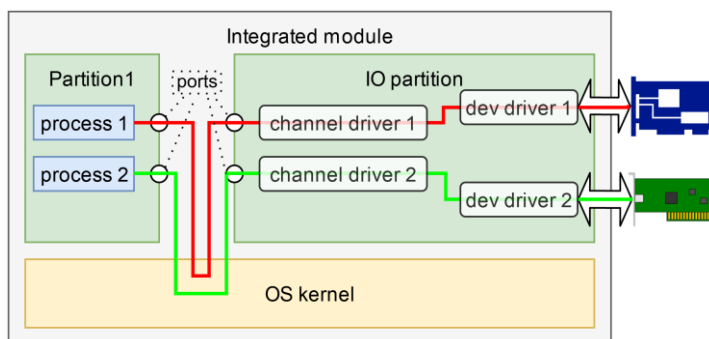

Fig.5. Two messages are being parallel sent to different network cards

Different drivers require different configuration. We have dedicated xml parsers of some specific part of xml document, this parser generates data specific for corresponding driver.

This architecture allows independent work of different drivers, which can possibly come from different developers. Furthermore, it allows adding new drivers with minimal effort and change of common parts.

5. Future work

There is research group to develop OpenGL renderer and frame buffer driver for our OS. Their work will show how well we thought out architecture of IO partition.

We finally need to measure latency without providing which we cannot tell that our operating system is a real-time system.

We are going to seek way to minimize kernel code, and move code, for which it is possible, to user-space.

Currently we use only one CPU core of the e500mc multicore processor. Newest version of ARINC 653 introduces interfaces for multicore work. We are going to support multicore CPUs as well.

Another objective is to port the OS to MIPS CPU family and another PowerPC family, namely IBM PPC 440.

6. Conclusion

In this paper, we sketched JetOS, a real-time operating system, which support ARINC 653 standard. Our system started as fork of POK OS. We describe architecture of POK, architecture of our operating system and differences between them.

References

[1]. Avionics application software standard interface part 0 overview of ARINC 653, ARINC specification 653P0-1, August 3, 2015

[2]. Avionics application software standard interface part 1 – required services, ARINC specification 653P1-3, November 15, 2010

[3]. G. Bloom, J. Sherrill. 2014. Scheduling and thread management with RTEMS. SIGBED Rev. 11, 1 (February 2014), 20-25. DOI=http://dx.doi.org/10.1145/2597457.2597459

[4]. C. S. Stangaciu, M. V. Micea, V. I. Cretu; Hard real-time execution environment extension for FreeRTOS Conference: IEEE International Symposium on RObotic and SEnsors Environments (ROSE 2014), At Timisoara DOI: 10.1109/ROSE.2014.6953035

[5]. VxWorks 653 http://www.windriver.com/products/product-overviews/PO_VxWorks653_Platform_0210.pdf

[6]. R. Kaiser, S. Wagner: Evolution of the PikeOS Microkernel, MIKES: 1st International Workshop on Microkernels for Embedded Systems. 2007

[7]. LynxOS http://www.lynx.com/products/real-time-operating-systems/lynxos-rtos/

[8]. M. Masmano, Y. Valiente, P. Balbastre, I. Ripoll, A. Crespo, J.J. Metge, 2010. LithOS: a ARINC-653 guest operating for XtratuM. In Proc. of the 12th Real-Time Linux Workshop, Nairobi (Kenya).

[9]. M. Masmano, I. Ripoll, A. Crespo, and J.J. Metge. XtratuM: a Hypervisor for Safety Critical Embedded Systems. 11th Real-Time Linux Workshop. Dresden. Germany. http://www.xtratum.org/files/xm_rtlw09.pdf

[10]. S. H. VanderLeest. ARINC 653 hypervisor. In Proc. Of IEEE/AIAA DASC, Oct. 2010.

[11]. J. Delange, L. Lec, 2011. POK, an ARINC653-compliant operating system released under the BSD license. In 13th Real-Time Linux Workshop (Vol. 10). http://julien.gunnm.org/data/publications/articledl11-osadl11.pdf

[12]. S. Han and H.-W. Jin. 2012. Kernel-level ARINC 653 partitioning for Linux. In Proceedings of the 27th Annual ACM Symposium on Applied Computing (SAC '12). ACM, New York, NY, USA, 1632-1637.
DOI=http://dx.doi.org/10.1145/2245276.2232037

Устройство и архитектура операционной системы реального времени

[1, 2] *К.М. Маллачиев <mallachiev@ispras.ru>*
[1, 2, 3] *Н.В. Пакулин <npak@ispras.ru>*
[1,2,3,4] *А.В. Хорошилов <khoroshilov@ispras.ru>*
[1] Институт системного программирования РАН,
109004, Россия, г. Москва, ул. А. Солженицына, д. 25.
[2] Московский государственный университет имени М.В. Ломоносова,
119991, Россия, Москва, Ленинские горы, д. 1.
[3]Московский физико-технический институт (государственный университет)
141701, Россия, Московская область, г. Долгопрудный, Институтский пер., 9
[4] Национальный исследовательский университет «Высшая школа экономики»
101000, Россия, Москва, ул. Мясницкая, д.20

Аннотация. Современные авиалайнеры, такие как Airbus A320, Boeing 787, перспективный отечественный самолёт МС-21, используют новую архитектуру построения комплекса бортового оборудования, получившую название Интегрированная модульная авионика (ИМА). В её основе лежит объединение приборов и бортовых вычислителей в единую сеть реального времени, что позволяет существенно снизить количество кабелей на борту и, тем самым, уменьшить взлётный вес лайнера. В ИМА разделяются функции сбора информации (датчики), воздействия (актуаторы) и логики оказания управляющих воздействий, которая реализуется специализированным прикладным ПО в бортовых вычислительных модулях. Международный стандарт ARINC 653 описывает требования к операционной системе реального времени, устанавливаемой на таких модулях, и программный интерфейс между прикладным авиационным ПО и операционной системой. Данный стандарт регламентирует временное и пространственное разделение прикладного ПО в соответствии с принципами ИМА. Большинство ОСРВ соответствующих стандарту ARINC 653 являются коммерческим ПО. В данной статье представляется JetOS – ОСРВ с открытым исходным кодом полностью соответствующую требованиям ARINC 653 части 1 версии 3. JetOS была основана на открытом проекте французских исследователей POK. Некогда POK была единственной ОСРВ с открытым исходным кодом, которая хоть сколько-нибудь соответствовала требованиям стандарта ARINC 653, однако была непригодна для практического использования: POK не удовлетворяла ряду фундаментальных требований ARINC 653 и работала только в эмуляторе. При разработке JetOS код POK был существенно переработан. В статье мы обсуждаем недостатки POK и показываем, как нам удалось решить эти проблемы и какие изменения были внесены в архитектуру и реализацию POK и отдельным подсистем. В частности, был полностью переписан планировщик реального времени, сетевой стек и управление памятью. Также в JetOS были добавлены новые возможности. Наиболее интересной является поддержка

системных разделов. Системный раздел – специальное прикладное ПО с расширенным набором возможностей, таких как прямой доступ к отдельным аппаратным средствам (сетевой карте, PCI контроллеру и т.п.). Наличие системных разделов позволяет вынести крупные подсистемы из ядра ОС и оставить в ядре минимальный набор задач, связанных с переключением контекстов, планировщиком и обменом сообщениями между компонентами ПО. В частности, в системный раздел вынесена подсистема, отвечающая за взаимодействие через сеть. Данное перемещение кода позволяет уменьшить размер ядра ОС, что теоретически уменьшает вероятность наличия ошибки в ядре и упрощает процесс верификации ядра.

Список литературы

[1]. Avionics application software standard interface part 0 overview of ARINC 653, ARINC specification 653P0-1, August 3, 2015
[2]. Avionics application software standard interface part 1 – required services, ARINC specification 653P1-3, November 15, 2010
[3]. G. Bloom, J. Sherrill. 2014. Scheduling and thread management with RTEMS. SIGBED Rev. 11, 1 (February 2014), 20-25. DOI=http://dx.doi.org/10.1145/2597457.2597459
[4]. C. S. Stangaciu, M. V. Micea, V. I. Cretu; Hard real-time execution environment extension for FreeRTOS Conference: IEEE International Symposium on RObotic and SEnsors Environments (ROSE 2014), At Timisoara DOI: 10.1109/ROSE.2014.6953035
[5]. VxWorks 653 http://www.windriver.com/products/product-overviews/PO_VxWorks653_Platform_0210.pdf
[6]. R. Kaiser, S. Wagner: Evolution of the PikeOS Microkernel, MIKES: 1st International Workshop on Microkernels for Embedded Systems. 2007
[7]. LynxOS http://www.lynx.com/products/real-time-operating-systems/lynxos-rtos/
[8]. M. Masmano, Y. Valiente, P. Balbastre, I. Ripoll, A. Crespo, J.J. Metge, 2010. LithOS: a ARINC-653 guest operating for XtratuM. In Proc. of the 12th Real-Time Linux Workshop, Nairobi (Kenya).
[9]. M. Masmano, I. Ripoll, A. Crespo, and J.J. Metge. XtratuM: a Hypervisor for Safety Critical Embedded Systems. 11th Real-Time Linux Workshop. Dresden. Germany. http://www.xtratum.org/files/xm_rtlw09.pdf
[10]. S. H. VanderLeest. ARINC 653 hypervisor. In Proc. Of IEEE/AIAA DASC, Oct. 2010.
[11]. J. Delange, L. Lec, 2011. POK, an ARINC653-compliant operating system released under the BSD license. In 13th Real-Time Linux Workshop (Vol. 10). http://julien.gunnm.org/data/publications/articledl11-osadl11.pdf
[12]. S. Han and H.-W. Jin. 2012. Kernel-level ARINC 653 partitioning for Linux. In Proceedings of the 27th Annual ACM Symposium on Applied Computing (SAC '12). ACM, New York, NY, USA, 1632-1637. DOI=http://dx.doi.org/10.1145/2245276.2232037