

# NevOS - The Scalable Real-Time OS for Embedded and Distributed Systems

*More and more companies are using a Real-Time Operating System (RTOS) to execute RT tasks. There is no RTOS that would be appropriate for any RT task. Every RTOS will suit only some classes of RT systems that should be defined for it at the development stage as well as at considering its application in a RT system project. The architecture of a RTOS, such feature as scalability influence a lot to the RTOS characteristics and application domains. The paper considers how the multilayered structure of the NevOS - a distributed RTOS, designed according to the proposed distributed RTOS modular architecture, makes it suitable both for small embedded systems and for large distributed systems.*

## INTRODUCTION

The RTOS design remains an up-to-date problem. The existing OS standards do not meet requirements for RTOSs. There are no generally-accepted standards for RTOS, there are no defined and accepted interfaces for the tools (compilers, debuggers, etc.), and there are no generally accepted standards for the I/O subsystem interfaces (drivers, graphics, etc.). The POSIX standard was discussed many times, (1, 2). Although there were many good things in POSIX that allow to support several classes of RTSs, in general, POSIX is too cumbersome, requires too much memory, has inappropriate syntax, or loses real-time response (performance) because of the bulkiness. In the RTOSs, that were developed under POSIX specifications, the top-down approach is used. Trying to add scalability and modularity to an existing large-system standard is difficult if not impossible. Therefore RTOSs must be developed from the bottom-up with scalability and modularity features. There are many other software standards besides POSIX. Most of these standards, however, have been developed either for medium systems or for large one with virtually unlimited resources or for embedded small systems.

The absence of the RTOS standard that meet up-to-date requirements provides confusion in the field of RTOSs and RTSs development. As a result, most RTOS projects are aimed to support the special RTSs.

It seems strange but the obstacle in up-to-date RTOS development is the absence of clear definitions of the basic notions of RTSs: their classes, hard and soft RTSs, RTOS classes and etc. These notions will allow to specify the requirements for RTOS and to design RTOS that takes into account RTS requirement, to develop general standards for RTOSs.

Before RTOS design we specify a set of notions. Based on them the requirements for up-to-date RTOS are defined. We propose the approach for the RTOS

design which takes into account RTS classes requirements and features and allows flexible modification of the RTOS structure according to these requirements. This approach is implemented in the NevOS - a distributed OS with real-time features.

## REAL-TIME TASKS AND REAL-TIME OPERATING SYSTEMS

To develop a high quality RTOS we should understand first of all what is a RTS what are its key features, what are the RTS's classes we are going to support.

Strange as this can seem, but there is no common formal definition of RTS that can give constructive view of RTS for a RTOS designer. The existing definitions either define real time in an informal way or reflect real-time features and requirements only partly, do not reflect features that are important for a designer of a RTOS or a computer system for RT tasks.

In general a RTS can be defined as a cortege of three elements:

In general a RTS can be defined as a cortege of three elements:

- **Task**  
a set of tasks which RTS has to perform;
- **Process**  
a set of processes that perform the RTS tasks;
- **Constrain**  
a set of time-constrains that RTS has to ensure.

A RTS properties are defined by the combination of task types, set of processes and time-constrains types.

There are four tasks types which defines the priority of "receiving" and "processing" functions of the task and the type of receiving and processing signals (data): separate signals or signals flow.

A RTS can consist of one or more processes: their granularity, dynamics, connections with the input and output signals are essential attributes of the RTS.

**RTOSs must be developed from the bottom-up with scalability and modularity features.**

There are three main types of time-constraints for a RTS:

- a time characteristic of RTS has to meet its deadline and should not miss it;
- a time characteristic of RTS can miss its deadline but the quality of the result is decreasing;
- a time characteristic mean of RTS cannot exceed deadline mean.

Most papers, dedicated to real-time problems, contain the similar hard and soft RTS definition: RTS is called "hard" if the requirement (time-constraint) is that the system has to respond to external or internal signals within a specified deadline, (3), and is called soft if the system has a deadline but can miss it. This definition is not full one for a RTS designer because it does not reflect the essential features of the RTOS. To choose a RTOS it is important to know not only whether we could or could not miss a deadline, but the nature of the constraints and their correlation with the characteristics of the computing entity in which the RTS is running, with the characteristics of the RTOS and the computer system.

For example, let us have two RTSs and each of them has to meet its deadline. The first one solves a task with the requirement: RTS has to receive each input signal (data) and generate an output for that signal within the defined time interval (this type of RTSs is typical for control systems). The requirement for the second RTS is to receive a flow of signals and generate a flow of outputs without missing any input (it is typical for signal processing). Of course first task is harder to support than the second one.

Second example. Let us have two RTSs which have to meet their deadlines. They solve the same tasks, which would be performed, say, in 50 ms in exclusive mode, but these two RTSs have different deadlines: the first — 100ms, the second — 10s. Of course we should call the first RTS as "hard" and the second as "soft". The RTS can be called a hard real-time RTS if time-constraints (requirements) are commensurable with the time that is necessary to execute the task in the exclusive mode. A RTS is a soft real-time RTS if time-constraints are much more than the time for the task execution in the exclusive mode.

A RTS performs its functions in the environment that is organized by RTOS. Three classes of RTOS can be picked out, (4). They are distinguished by the computing environment which they organize and support.

A Local RTOS class provides multiprocess environment inside a uniprocessor or in one node of a multiprocessor system. For instance, the RT-Kernel, (5), belongs to this class.

A Network RTOS provides a multiprocess environment in a distributed computer system and support access to the remote resources of the network. A user of a Network RTOS must know the network structure and the resources of each node and explicitly control the resource distribution and access for the processes of

the RTS. Such RTOSs as VxWorks, (6), and QNX, (1), are the Network ones.

A Distributed RTOS provides a "virtual single processor", (7), with transparent distributed environment. A user does not know about features of the network and its structure. Such RTOSs as Chorus, (8), RTSM, (9), and RTX/MP, (7), belong to the distributed RTOS class.

The choice of a RTOS class to execute the RTS's tasks and should be based on the advantages, drawbacks and functionality of a RTOS class and its consistency with the RTS constraints.

The distributed RTOS structure depends upon features and constraints of RTS from above and computer system architecture features from below. The main objective of a RTOS is to provide computing environment for RT systems' processes. The RTS requirements have effect on the RTOS's ones. Each RTS requirement affects on a set of RTOS requirements (both qualitative and quantitative ones). We can formulate general requirements for a distributed RTOS, depending upon the RTS features and time-constraints.

- A RTOS has to be multiprocess and perceptible. It means not that a distributed RTOS should be fast, but that the maximum time to do something should be known in advance and should be consistent with the RTS requirements.
- A RTOS should be predictable. The time characteristics of system calls should be known, and must be consistent with RTS requirements.
- A RTOS has to support predictable mechanisms of interprocess communications and synchronization for processes resided both inside one node and in different nodes of distributed computer system.
- A RTOS has to support process priority and process binding with time and external events.
- A distributed RTOS must have a load balancing mechanism for effective usage of distributed computer system resources.

It would be useful to design a RTOS which would satisfy all these requirements and could be used both for uniprocessor embedded RTS and for distributed RTSs.

## HISTORICAL BACKGROUND

The RTOS NevOS in its concepts and evolution is the result of integration of the theoretical background in parallel and distributed processing and the practical experience of real-time programming for embedded systems in avionics, telecommunications, control systems, etc.

The works in parallel architectures and computing formed the general model of a computer system (uniprocessor, multiprocessor, distributed, mass-parallel) — a virtual machine with dynamic architecture, that is coincided with the structure of a program that

**The choice of a RTOS class to execute the RTS's tasks and should be based on the advantages, drawbacks and functionality of a RTOS class and its consistency with the RTS constraints.**

is running in it, (10). There was developed a model of parallel computations — the "Asynchronous Developing Processes" (ADP), that presents a running program as a dynamic network of processes. The ADP belongs to the class of asynchronous models of parallel computations, data-flow computations can be considered as its particular case. The ADP model follows the concept of separation of the parallel program schema and the interpretation of its objects — processes (actors) and data-objects. The control of the program flow in a program is presented (and specified) at the level of the program's schema and is reflected in its continues self-modification. The ADP model defines the rules for constructing the semi-interpreted program schema and their operation. By constructing a program schema and giving an interpretation of all its objects according to the task we get the fully interpreted multiprocess program's schema — that is the program. These concepts were implemented in several projects of experimental parallel computers with distributed architecture, (11, 12, 13,14).

The other predecessors of NevOS were the real-time executives that we developed for various projects in avionics, telecommunications, (15), and control systems. They were included in the application program packages as their integral com-

ponents.

Naturally, any real-time program is a network of processes and requires a multiprocess entity to operate in. A correct multiprocess schema of the program is of vital importance for the quality and reliability of the program. We believe that there is nothing worse for it than spreading the operators, that lay out a program's schema, over the sequential programs of individual processes. The RTOSs we had considered just followed this wrong concept of forming the multiprocess programs. Our practice corroborated this point of view. We developed our own RT-executives. Their application in particular projects contributed a lot in timely and sound outcomes.

---

**The three layer architecture of distributed RTOS with microkernel and subsystems allows to satisfy requirements of a wide variety of RTSS**

Along with the RT executives and parallel software we developed the programming tools for them: debuggers (e.g. the distributed debugger "ROM", (16)) and parallel programming environments, including the language and programming toolset VISA for visual parallel and distributed programming, (17).

The real-time processing has much in common with the parallel and distributed computing. In both fields the core is organising the computations as ensemble of processes working in parallel

(really or virtually). The paradigm of a computer system as a virtual machine with dynamic architecture, that forms an entity for running networks of parallel processes, gives a general guideline for their integration. The main point was to ensure the consistency between the real-time notions and constraints with the asynchronous nature of distributed parallel computing in such models as the ADP. We managed to do it and to implement in the operating system called NevOS.

The NevOS was developed as a real-time OS with features for distributed processing support. Though it was designed as mobile RT OS for various microprocessors up to now it has an industrial implementation for the iX86 microcomputer platforms. It is used in projects of embedded and control systems in avionics, telecommunications and control systems, (18,19), as well as for experiments in parallel computing in distributed architectures.

## NEVOS ARCHITECTURE

The three layer architecture of distributed RTOS with microkernel and subsystems allows to satisfy requirements of a wide variety of RTSs, from embedded uniprocessor RTSs to distributed RTSs with large number of processes.

NevOS was designed following multilayer hierarchical approach, (20).

The first (the lowest) one is the Local OS layer. It is the layer of uniprocessor multitasking RTOSs which organise multiprocess environment in the individual nodes of a heterogeneous distributed computer system. The main features of this layer are source code portability and flexible structure of RTOS. The microkernel RTOS architecture allows to satisfy these requirements.

The Local OS layer of NevOS contains kernel and systems. NevOS kernel contains the main system functions of RTOS. It allows to minimise the kernel size and use it for small embedded control systems. The minimal Local OS layer of NevOS consists of only the kernel.

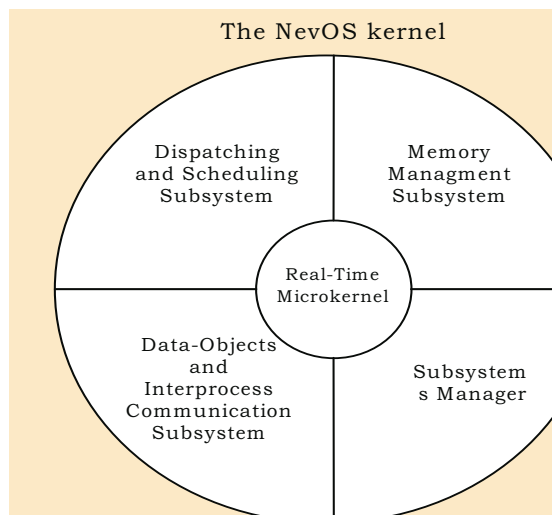


Figure 1. The NevOS Kernel

The NevOS kernel consists of the microkernel and kernel subsystems. The microkernel is hardware-dependent. It allows to use all hardware features to improve RTOS characteristics. To provide NevOS mobility only the microkernel depends upon the hardware features. The other parts of the kernel (kernel subsystems) are mobile and use microkernel to execute kernel functions (fig.1) that can be hardware-dependent.

The NevOS microkernel is compact and implements the basic set of services:

- process creation and termination;
- process priority scheduling;
- memory management;
- interrupt dispatching;
- interprocess communications primitives;
- timer management;
- real-time clock.

NevOS microkernel and four main subsystems: dispatching and scheduling subsystem, memory management subsystem, interprocess communication subsystem and subsystems manager, constitute the NevOS Kernel.

**The microkernel is hardware-dependent. It allows to use all hardware features to improve RTOS characteristics.**

A system in NevOS is a set of system processes, system Data-Objects and/ or functions that are included in the kernel. The kernel's functions are grouped in the kernel's subsystems. They constitute the system program context of a process, running in NevOS.

In the terms of the services provided the kernel of NevOS is comparable to a small real-time executive but supports scalability of NevOS through its kernel and systems.

The Network layer of NevOS is formed by assembling the Local OSs of the distributed computer system nodes by the Transport subsystem (TS). TS provides services for remote process communication and ensures access to the remote resources. It accounts the distributed over nodes resources, forms the links between the distributed processes. TS is used by the application processes of the NevOS network layer and by the system processes of the NevOS Distributed Virtual Computer layer.

The Transport System is presented to its users - system and application processes - by its interface specification. The Transport Subsystem Interface (TSI) specification for the NevOS is oriented towards the dynamic organisation of network of processes in the reconfigurable distributed computer system with special features for real time systems support, (21).

The main task of the Transport System is to provide efficient, reliable, and cost-effective data transport from a process at the source node to a process(es) at destination node(s), independently of the used physical network. The Transport System includes a number of transport entities and the transport protocol for their interaction. The interaction between transport entities is supported by the stack of protocols

(standard ones where it applicable, original where is not) and hardware facilities that set up the communication network of a distributed computer system. The Transport subsystem provides services for the other distributed subsystems of NevOS and is operated by systems calls. In its turn, the Transport subsystem uses the services provided by the low-layer protocols, but it is transparent for the users of the Transport system.

The processes that located in the separate nodes of the distributed computer system are connected by the Multipoint Transport Channels (MTCs). The MTC is generated and destroyed by the Transport system directives. Within the MTC one can organise the MULTICAST, BROADCAST, and, of course, UNICAST (point-to-point) message transmissions.

Besides usual means for message transfer the Transport system of NevOS has the distributed computation support, (21).

The Transport system of NevOS has to take into account some specific features and requirements for the RTS support, such as interrupt distribution, access to real-time clock, and guaranteed predictable time of message transmission. These RT-specific features are reflected in the special set of the Transport Subsystem calls and in appropriate mechanisms for real time in general purpose calls.

The third layer — the **Distributed Virtual Computer** layer — provides system functioning as a single computer with an integrated resource pool. This layer controls process allocation, execution, termination in the distributed computer system as in integral computing entity, executes load balancing and provides fault-tolerance implicitly for RTS processes. The main requirement for this layer is to provide effective dispatching of distributed computer system resources ensuring predictable time of execution and interaction of a RT program objects.

The Distributed Virtual Computer layer is formed by adding to the Network layer the systems which control the distributed resources; allocate processes; provide load balancing. This layer organises distributed parallel processing and should follow an appropriate model of parallel computations with distributed control and real-time notions included.

The Distributed Virtual Computer layer forms a multi-process computer entity with transparent multi-processor organisation for the RTSs. An application RTS process runs in this entity without connection with any number or position of a node in the computer system.

Unlike the pure parallel processing, a RTS in general, runs in the heterogeneous computer system. For instance, some nodes of the computer systems have inputs or outputs to/from the object of control, the other do not have it. At the layer of Distributed Virtual Computer the transparency of the internal computer system organisation must be combined with the insight into its heterogeneous nature. In NevOS this problem is solved by including the notion of resource type in the Application Program Interface (API). A process is bound not with a number or position of a

node, but with the types of resources it requires. The scheduling and allocation of processes over the nodes of the computer system is left to the operating system and is transparent for the application program.

A program for NevOS is a dynamic network of processes (RTS processes) which communicate via special objects — Data Objects. There are two types of processes: interrupt handlers, which are controlled by hardware dispatcher, and processes, which are controlled by NevOS's dispatcher. To satisfy RT requirements NevOS supports a special type of interrupt handler: interrupt handler with watch-dog timer. There is special subsystem in NevOS which controls timers and provides real-time clock access.

The special forms of the message-passing mechanism are used for interprocess communications. Processes send and receive messages through Data-Objects without mutual synchronisation of the processes. A process reads a message from a Data-Object if it is not empty or waits if empty, and the process writes the message to the Data-Object if it is empty or waits if any.

## CONCLUSIONS

We have presented the multilayer hierarchical approach for a RTOS design. Based on this approach RTOSs that satisfies requirements and provides features of various RTS classes can be developed.

The key feature of the approach is scalability and modularity of the RTOS structure. For the Network and Distributed Virtual Computer layers a model of distributed computations with real time features, protocol stack and transport provider interface are most important. This approach allows to build the RTOS from small embedded uniprocessor one to large distributed RTOS satisfying requirements of a particular RTS class.

The successful implementation of some embedded RTSs using multilayer approach proves the its viability for RTSs. The flexible scalable RTOS NevOS in its basics is implemented and used in various applications. It is further developed for prospective distributed high-performance real-time processing (e.g. over ATM).

## REFERENCES

1. Hildebrand D. A Microkernel POSIX OS for Real-Time Embedded Systems. Real-Time Magazine. Issue 1993/3. P.82.
2. The ESSE - The Embedded System Software Environment./ VITA Project. 1995 (<http://www.vita.com/esse/esseindex.html>).
3. Silly M. A Dynamic Scheduling Algorithm for Semi-Hard Real-Time Environments. - In: Proceedings of the 6th Euromicro Workshop on Real-Time Systems. Vaesteraas, Sweden. June 15-17, 1994. PP. 130-137.
4. Nedzelskaya N.D. Distributed Operating Systems for Distributed Real-Time Systems Control - In: Proceedings of the Conference "Diagnostics,

- Infomatics, Metrology, Ecology, Safety- 96". St.Petersburg, 1996. PP. 194-195. (In Russian).
5. RT-Kernel Real-Time Multitasking Kernel 4.0/ Informatik GmbH, 1995. 8p.
  6. VxWorks: Programmer's Guide Release 5.0. Wind River Systems Inc. 1992. 334p.
  7. RTXC/MP: Virtual Single Processor Real Time Kernel. Intelligent Systems International, Linden, 1991. 9p.
  8. CHORUS on H1: UNIX System V on Networks of Transputers/ D.Grabas, M.Guillemont, M.Tombroff et al.// Transputing'91/ Ed. by P.Welch et al. IOS Press. 1991. P. 262-280.
  9. RTSM: Real-Time Operating Systems. Parsytec, Aachen, 1991. 478p.
  10. Mjasnikov V.A., Ignatiev M.B., Sheinin Y.E. Multimicroprocessor Computer System Architecture// Cybernetics. 1984. N 3. P. 48 - 55. (In Russian).
  11. Ignatiev M.B., Mjasnikov V.A., Sheinin Y.E. System 3M - the Experimental Multiprocessor System// Proc. of the 2nd Symposium on Microcomputer and Microprocessor Applications OMKDK/ Technoinform, Budapest. 1981. (In Russian).
  12. Ignatiev M.B., Sheinin Y.E. Recursive Computers and New Generation Computers// New Generation Computers: Architecture, Programming, Intelligence: Proc./ Vc SO AN SSSR. Novosibirsk 1986. P.27 - 38. (In Russian).
  14. Distributed Computing and VLSI Systems: Proc./ Ed. by M.B.Ignatiev, Y.E.Sheinin; LIAP. L., 1988. (In Russian).
  15. Sheinin Y.E. Recursive multimicroprocessor computers // Multimicroprocessor Systems: Proceedings of the 3rd Symp./Ed. by D. Hammer. Berlin, 1989 P. 127 - 137.
  16. Gorbachev S.V., Samoylenko S.I., Solonina N.B., Ushkov V.I., Sheinin Y.E. The Adaptive Packet Switching Transputer Centre for Highspeed Networks// Proceedings of the II International Conference on Regional Informatics, SPb, 1993. (In Russian).
  17. Aladova T.E., Ignatiev M.B., Sheinin Y.E. Distributed Monitor for Multimicroprocessor Systems Software Debugging. // Microprocessor Devices and Systems. 1990. N 5. P.49 - 55. (In Russian).
  18. Ignatiev M.B., Sheinin Y.E., Tatkov D.E. VISA - The Interactive Visual Language for Parallel and Distributed Programming// Supercomputing'96 Conference. Proc./Pittsburgh. 1996.
  19. Aladova T.E., Nedzelskaya N.D., Sheinin Y.E. Multiprogram Operating System for Microcomputer Platforms and Distributed Systems. //The 2nd Conference on the Problems of Acquisition, Processing and Transmission in Complex Radioelectronic Systems. /VURE PVO, Pushkin, 1995, v.1.
  19. Fragin M.S., Gorbachev S.V., Ignatiev M.B., Pomelov S.Y., Sheinin Y.E. The Modular Multimicroprocessor System Architecture for the Power Turbine Control //Proceedings of Sixth EUROMICRO Workshop on Real-time Systems. Vaesteraas, Sweden/ IEEE Computer Society Press. Los Alamitos. 1994. P.124-127.
  20. Aladova T.E., Nedzelskaya N.D., Sheinin Y.E. The Distributed Operating System with Real Time Features. - In: Proceedings of the 5th International Conference "Distributed Information Processing - DIP-95". Novosibirsk. 1995. (In Russian).
  21. Verenich A.I., Gorbachev S.V., Sheinin Y.E. The Transport System Interface Specification for Parallel Distributed Computer Systems. - In: Proceedings of the 6th Conference "Transputer Systems and Their Application". Domodedovo. 1996. P. 21. (In Russian).

*Tatiana Aladovais is Senior Researcher, the head of System Programming sector in Microprocessing Technologies. She graduated from the Leningrad Institute of Avionics, 1980. She was research fellow in the State Academy for Aerospace Instrumentation (1986-1991), senior engineer (1983-1985), engineer in the Leningrad Institute of Avionics.*

*Dmitriy Mikhailichenko is an engineer. He graduated from the State Academy for Aerospace Instrumentation, 1997.*

*Natalia Nedselskaya is a post-graduate student for Ph.D in software engineering. She got the Master of Science degree in computers from the State Academy for Aerospace Instrumentation, 1994.*

*Yuriy Sheinin is the director of the Microprocessing Technologies company. He got the Master of Science degree in computers, 1973, Ph.D in Computer Science, 1979, Leningrad Institute of Avionics. He was the Head of the Microprocessor Systems research department in the State Academy for Aerospace Instrumentation, (1986-1990), Head of the research laboratory (1980-1985), Research fellow (1976-1979), Senior Engineer (1974-1975), Engineer (1973-1974) in the Leningrad Institute for Avionics, USSR, assistant professor, lecturing high performance computing and networking, supervising post-graduate students in these fields (1985 to present) in the State Academy for Aerospace Instrumentation, St.-Petersburg, RUSSIA.*

## Real-Time on the Net

