UDC 681.3.066

Turkin I., Nikitina T.
The National Aerospace University "KhAI",
Ukraine

MODERN CONCEPTS OF REALIZATION FUNCTIONAL SAFETY AND RELIABILITY IN REAL-TIME OPERATING SYSTEMS

In the given research are analyzed methods of reliability and safety of Real-Time Operating Systems (RTOS). Standards of safety for RTOS and existing developments in the given direction are discussed in detail, also the concept of realization partitioning Operating System, RTOS architecture are viewed. Realization of key mechanisms of work with virtual memory, interaction of processes and threads according to POSIX are explained.

At present many of important government objects are under the control of unreliable operational systems MS Windows and MS DOS. Some objects use existing operational systems on the basis of operational system (OS) Unix, however and it does not give necessary safety, and also these systems are too bulky and exacting to resources to use them on military-industrial objects - for example in aircraft. Abroad had already created some systems which are applied in the crucial objects demanding maximal reliability and safety. So, for example, system Tornado/vxWorks is used on mars rovers Spirit and Opportunity, and also on some air liners, system LynxOS is used in aircraft, on fleet, on modern tank machines. The broadest application finds OS QNX. All mentioned RTOS has high degree of reliability and safety. Two principal causes compel to offer own development as alternative:

1. In systems of critical application closeness of an initial code creates potential danger.
2. Specific element base.

For development reliable and safe RTOS it is necessary to observe a lot of requirements: the system should work in real time, must be safe, reliable, functional; must consume a minimum of hardware resources; should be capable to expansibility (modularity), the initial code should be open; should meet the requirements of modern standards of safety and functionality. For guarantee of work in real time RTOS must give the response to any unpredictable external influences during a predicted interval of time. Developing RTOS should belong to a class of exacting real-time systems and observe the following principles:

- . Any delay is not comprehensible, under no circumstances.
- The result given with delay - is useless.
- Time deadline disturbance of the response is considered as catastrophic refusal.
- The price of time excess of the response is indefinitely high.

For further realization it is necessary to keep key rules and standards for UNIX-like systems, such as POSIX. Mechanisms of realization of safety, functionality and real-time requirements should satisfy the international standards in branches where given RTOS will be used. For example, now the following basic standards for RTOS are determined in the field of aircraft:

[7] ARINC 653 (Avionics Application Software Standard Interface) it is developed by company ARINC in 1997. This standard determines universal program interface APEX (Application/Executive) between OS of an aviation computer and application software.

[8] CCITSE (Common Criteria for Information Technology Security Evaluation) is a set of requirements and the conditions of privacy approved by Agency of national safety and National institute of standards and technologies in the USA. CCITSE determines assurance level of privacy - EAL (Evaluation Assurance Level). Common Criteria estimates not only safety and reliability of products, but also processes of their development and support.

[9] MILS (Multiple Independent Levels of Security/Safety) makes possible mathematical verification of a program kernel.

[10] POSIX (the Portable Operating System Interface) is a family of standards designed to ensure source-code portability of application programs across hardware and operating systems. POSIX was developed by the Institute of Electrical and Electronics Engineers (IEEE) and is recognized by the International Organization for Standardization (ISO) and American National Standards Institute (ANSI). The POSIX standards provide for communication between an application and the underlying operating system.

Because POSIX conformance ensures code portability between systems, it is increasingly mandated for commercial applications and government contracts.

For the developing RTOS will be used standard ARINC 653, for realization of the partition and standard POSIX, for internal realization of system (processes, threads and other).

For today among all existing RTOS, LynxOS supports a maximum quantity of standards of safety and reliability for devices and technologies with critical requirements - such as air liners, satellites, the military ships, and others. Therefore for realization RTOS it is offered to take for a basis mechanisms and concepts of this OS. OS achieves system security through Virtual Machine (partitions). Each RTOS partition performs like a stand-alone real-time operating system. System events in one RTOS partition can neither share resources nor interfere with events in another RTOS partition. OS partitioning involves exclusive access of three kinds: time, memory and resources. Time partitioning is done through a fixed-cyclic time-slice scheduler, which allocates periods of time to each partition. During each time slice, only processes in the assigned partition are permitted to execute. The OS will implements an ARINC 653-1-based time partition scheduling algorithm that gives each partition fixed execution time so that the system can be deterministically safe. Memory partitioning is achieved by dividing RAM into discrete blocks of nonoverlapping physical address space. Each RTOS partition is assigned one and only one block of memory. Within the partition, the virtual address spaces of various processes are mapped to memory from the assigned memory block. Resource partitioning means that each device can be assigned to only one partition of the RTOS. This means that a fault in a device or its driver will be contained within a single RTOS partition. Each partition mounts a RAM-based file system for data storage. The file systems are private to the individual partitions and are never shared with other partitions. RTOS supports a multiprocess, multithreaded environment in which real-time applications can run seamlessly, make system calls, and use device drivers. Figure 1 shows the RTOS architecture.

Common Initialization (cinit) is the first POSIX process to run after the kernel is initialized. Cinit executes with operating system root privileges. It reads the Virtual Machine Configuration Table (VCT) and creates VM partitions within OS.

The primary responsibilities of cinit are as follows:

- Validate and read the VCT.
- Load Device Drivers.
- Initialize system wide environment variables.
- Mount the file systems.
- Initialize the scheduler.
- Respond to partition fatal errors as defined in the VCT.

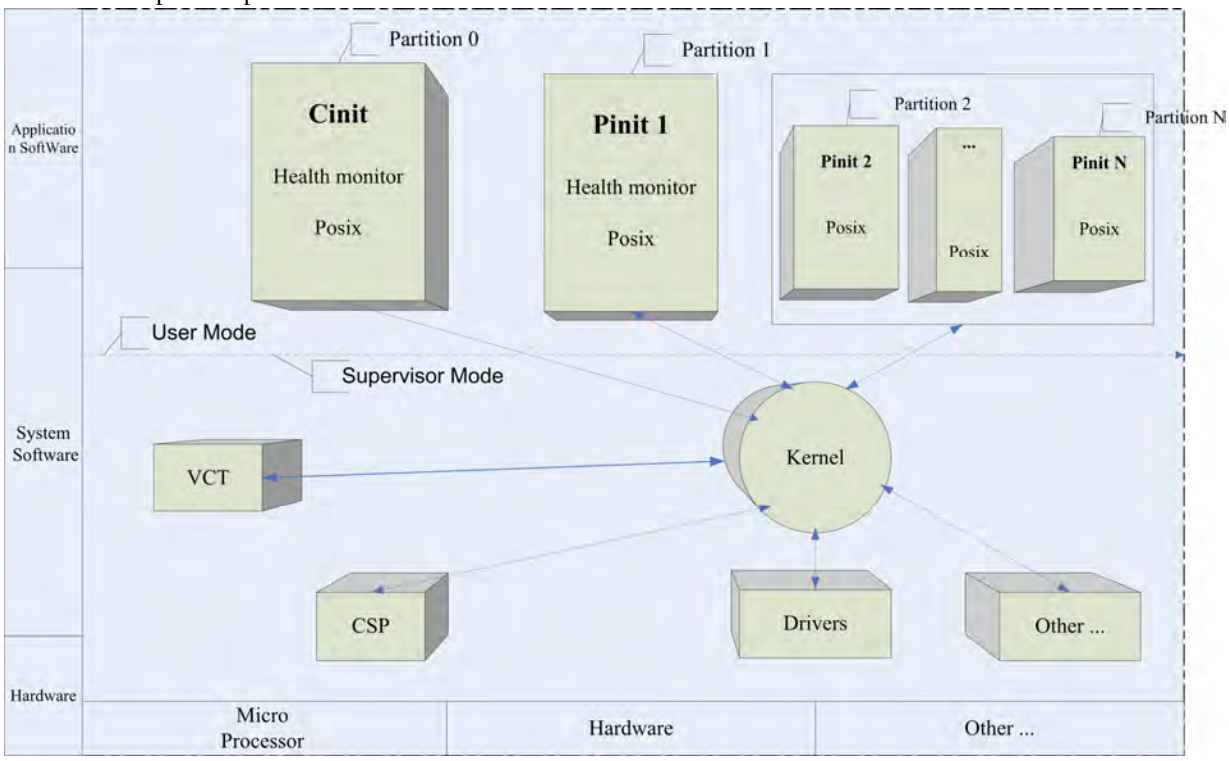

Fig. 1 RTOS Architecture

Partition Initialization (Pinit). Cinit transforms into a unique Pinit process in each partition. Pinit, as the first process in the partition, completes initialization of the partition's environment and transforms into the Application Software for the partition. Pinit executes with operating system root privileges.

Figure 2 shows t**hree processes mapped by the** Memory Management Unit (**MMU) into separate physical memory pages.** Each process is virtually isolated from other processes and it is therefore impossible for one process to write over another process' address space.

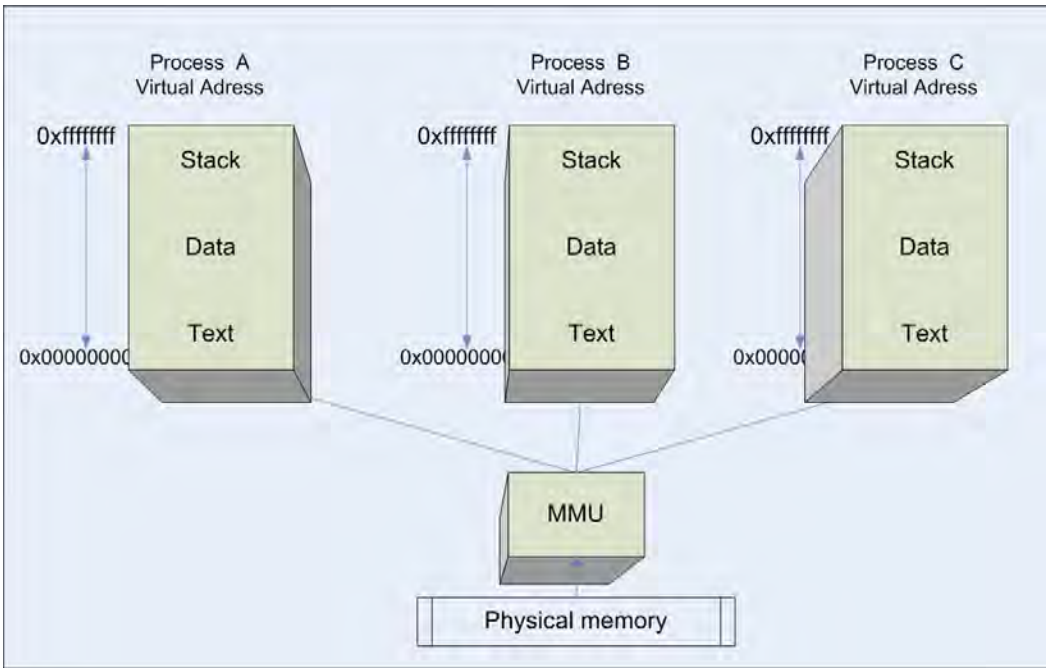

Fig.2. Three processes mapped by the MMU into separate physical memory pages.

The MMU is used to physically isolate processes from each another so that they cannot trample on each other's memory. Also commonly referred to as "multiprocessing," a process as defined by POSIX.1 will have its own name space. Two symbols defined globally will not conflict if they are located in two different processes, since they are in two different links. Processes and virtual memory:

1. Each process is assigned its own private memory in RAM. These memory locations may be anywhere in RAM.
2. The addresses used within programs are virtual addresses, which are translated at run time to indicate the real memory locations.
3. The translations are private to a process, so that only the process that owns the memory locations can access them. This insures that each process can only see and change its own memory, and cannot damage the memory owned by another process.

The set of virtual addresses that a process can access are called the process's virtual address space. A process virtual address space is composed of segments:

1. The text segment contains the program instructions.
2. The data segment contains global data and dynamically allocated data.
3. The stack segment contains the stacks owned by each thread.
4. Additional segments that are defined later.

The MMU translates the virtual addresses into physical addresses. If a process attempts to address a page that is not currently mapped to it, the MMU generates an exception. The exception sends a signal to the offending thread in which the default action is to terminate the process. Alteratively, the thread may catch the signal for user-defined actions, if desired. An important POSIX concept is the distinction between "threads" and "processes".

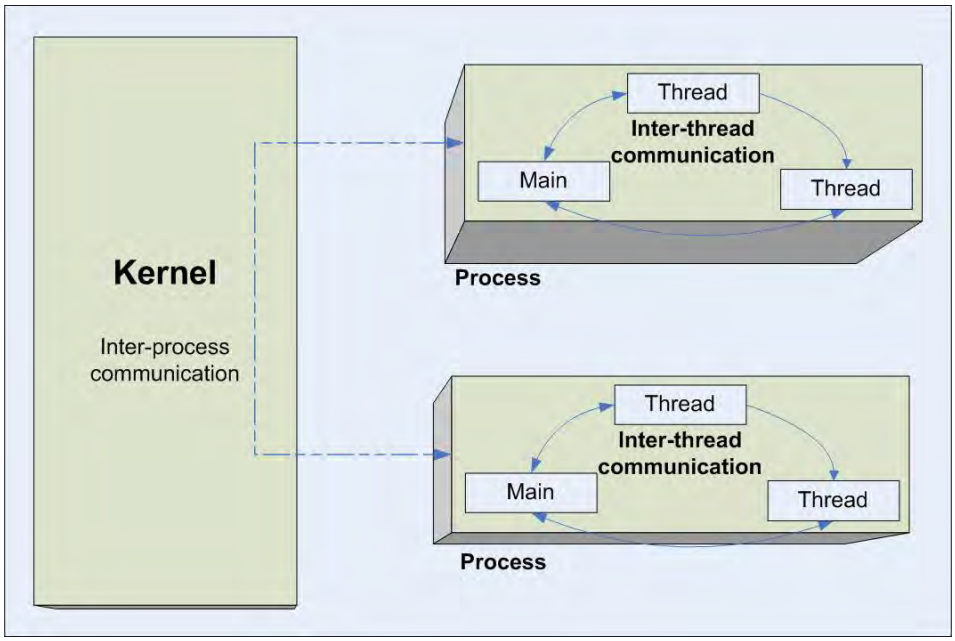

Fig.3. Distinction between "threads" and "processes".

Figure 3 shows distinction between "threads" and "processes". Threads are schedulable entities that run within a process. Each process will have one main thread, and it may also have several more threads which share the address space of the process. Note that some operating systems simply refer to "tasks" instead of distinguishing between "threads" and "processes." Such an operating system probably does not support the POSIX standard, and would not be able to run calls which refer to a process (such as signals). How POSIX threads work:

- A thread is a flow-of-control that runs within a process context.
- Threads in the same process share the same virtual address space.
- They can communicate and share data using globals.
- Threads do not have a parent-child relationship with each other.
- Any thread created within a process can terminate itself or can read the exit status of any other thread.
- Each process has its own protected address space.
- Communications between processes require kernel services.

Generalizing the resulted research and the practical side of development, it is possible to note, that portioning OS, partitions in RTOS were investigated, the realization of the given mechanism has been offered, standards of safety for RTOS are briefly considered.