Embedded Operating Systems for Microprocessor Architecture MIPS32

S. Osmolovsky
Saint Petersburg State University of Aerospace Instrumentation
Bolshaya Morskaya st., 67, 190 000, Saint Petersburg, Russia
serge.osmolovsky@gmail.com

Abstract

Great progress of the microelectronic industry for the last 15 years has caused a rapid growth in the field of embedded systems and has given a powerful impulse to the development of software for them, including the embedded operating system (OS). The paper presents methods and algorithms for designing a compact and reliable embedded operating system with a high response rate to events. Also this paper compares and analyzes the main characteristics of world market leaders in the field of embedded operating systems with support of MIPS32 microprocessor architecture and own developed software.

Index Terms: embedded OS, MIPS32, preemptive multitasking.

I. Introduction

Now such field of electronics as embedded systems is developing rapidly. Primarily it is connected with significant advances in technology and the rapid reduction in the cost components. Nowadays one of the most popular high-performance microprocessor architectures used in embedded systems is MIPS.

Operating systems in embedded systems provides an application standard interface (platform-independent). Embedded operating systems are usually fully integrated into a device and solve a specific tasks, for example, thread management, synchronization mechanisms between them and memory.

They must also possess:

- preemptive multitasking;
- support mechanisms of synchronization;
- priority for threads and the mechanism of priorities inheritance;
- interface with peripheral;
- small amount of code;
- predicted behavior.

At present in the world market there is a considerable quantity of commercial embedded operating systems that support MIPS32 architecture: QNX Neutrino (QNX), LynxOS-178 (LynxOS), Integrity, VxWork 653 (VxWorks), Inferno, Nucleus, RTLinux, eCOS, uC/OS-II, RTEMS and others. This paper provides an overview of the listed-above embedded OS based on public source information and comparing with characteristics of own developed software for the Multicore microprocessors (MC-12, MC-24). The central processor of the Multicore microprocessor has an architecture MIPS32 Little Endian.

II. Developing Embedded OS

Compact and reliable embedded OS must be constructed in a modular principle and must be well-scalable. Such embedded operating system can be based on microkernel (base module), which provides planning and scheduling of threads, synchronization between them using mutexes and messages, and interrupt handling. This implementation of kernel functions in developed embedded OS can improve overall system performance, because small microkernel can fit in processor cache. Other necessary modules (for example, device drivers and network protocols) can be built around the base kernel module. Developed embedded operating system must provide a set of API, in which there are functions for thread management, synchronization and organization of the interaction between them and the management of dynamic memory. Own software has been developed according to the basic design requirements of embedded OS. Source code of developed software has been written in high level programming language C. The assembler language is only used for machine-dependent features (for example, context switches and initialization of the stack). The interval timer of the Multicore microprocessors has been used to implement the functions of time management in developed software and mcLinux (scalable modular operating system with support MIPS32 microprocessor architecture and ported to a family of microprocessors Multicore).

A. Threads

The thread in embedded OS is part of the code from the programmer's viewpoint run concurrently with other threads, what is provided by separation of CPU time between them. Each thread has unique resources (processor registers, stack pointer). These resources are called the thread context. The thread has a separate stack, which is used for the storage of return addresses, parameters of function calls and context. Also, the thread has a priority and identifier.

Usually the thread can be in one of five states:

- dormant;
- running;
- ready;
- waiting;
- terminated.

The graph of transition between states is represented on Figure 1.

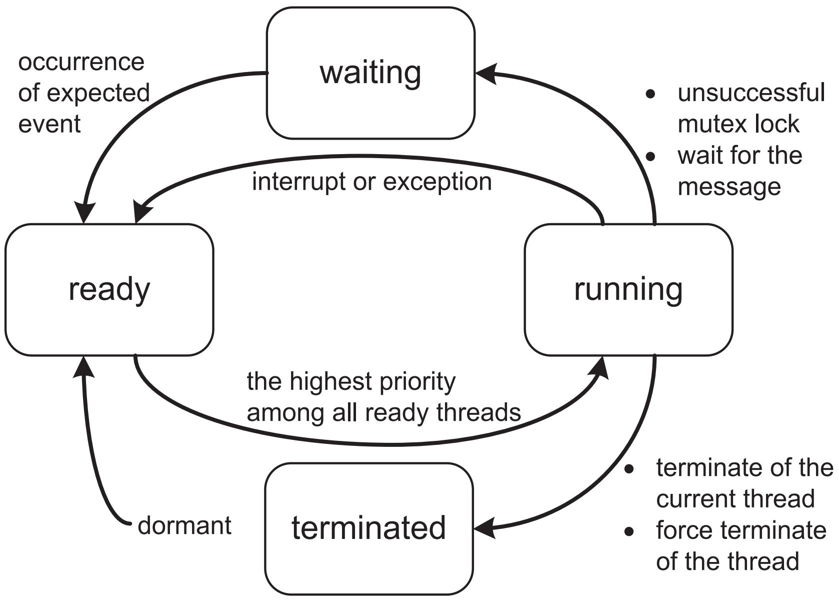
waiting
occurrence of expected event
unsuccessful mutex lock
wait for the message
interrupt or exception
ready
running
the highest priority among all ready threads
terminate of the current thread
force terminate of the thread
dormant
terminated

Figure 1. Thread states

During the execution of the thread it can capture necessary mutexes. Mutex is a object of synchronization, what is adjusted in a special signal state when it is free. Hence, mutexes provide correct access to resources from several threads.

B. Scheduling

Such embedded OS usually uses a dynamic scheduling algorithm [2]. The thread with the highest priority runs first, one after another. Priority is set at creation of a thread and can be changed at run time. If there are some threads with the same priority, then they will be controlled under the principle of FIFO, that is, on the sequence of being in the list of threads ready to execution.

Reliable embedded OS with predicted behavior has a preemptive multitasking, in which the system will place under control from one running thread to another in case of exceptions and interrupts, or the proceeds of certain messages from one thread to another. Processor time allocation is carried out by a thread scheduler who produces switching between threads, choosing the thread with the highest priority for execution, as shown in Figure 2.

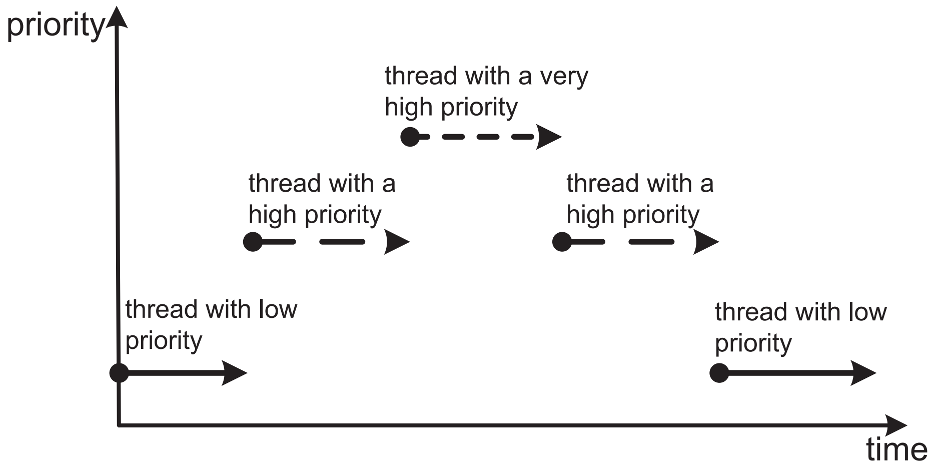
priority
thread with a very high priority
thread with a high priority
thread with a high priority
thread with low priority
thread with low priority
time

Figure 2. Preemptive multitasking

III. COMPARING EMBEDDED OS

One of the main requirements for an embedded operating system is a low-latency handling of an event. In practice it means that there must be small parameters such as interrupt latency and context switch time. It is not easy to objectively compare these parameters for various operating systems, since these parameters depend not only on the quality of the OS, but also on the capabilities of the hardware platform.

Debug complex MC-24EM SpaceWire is used for measuring and comparing the time parameters own developed software and mcLinux. The frequency of RISC-core was 100 MHz. It is measured the following time characteristics:

- context switch;
- interrupt latency (ISR and IST);
- scheduling latency.

For comparing of own developed software's parameters with other embedded OS, we used data obtained with tests of considered OS on different hardware with MIPS32 architecture. The test composition and the frequency of the microprocessor somewhat differs. This data were obtained on the basis of our tests and tests of our colleagues, tests by experts of magazine "Dedicated System" and other information from useful sources. According to these data the time of a context switch is in the range 5-50 microseconds and the interrupt latency - 2-30 microseconds. With the chart comparing the time characteristics of embedded operating systems, shown in Figure 3, we can conclude that own developed software has quite a high response rate to events in comparison with other operating systems. A LynxOS, compared with VxWorks and QNX, has a large interrupt latency (approximately in 2 times). Context switch time (switching time between two threads) is somewhat larger than one of QNX and less than about in 1.3 times compared with VxWorks.

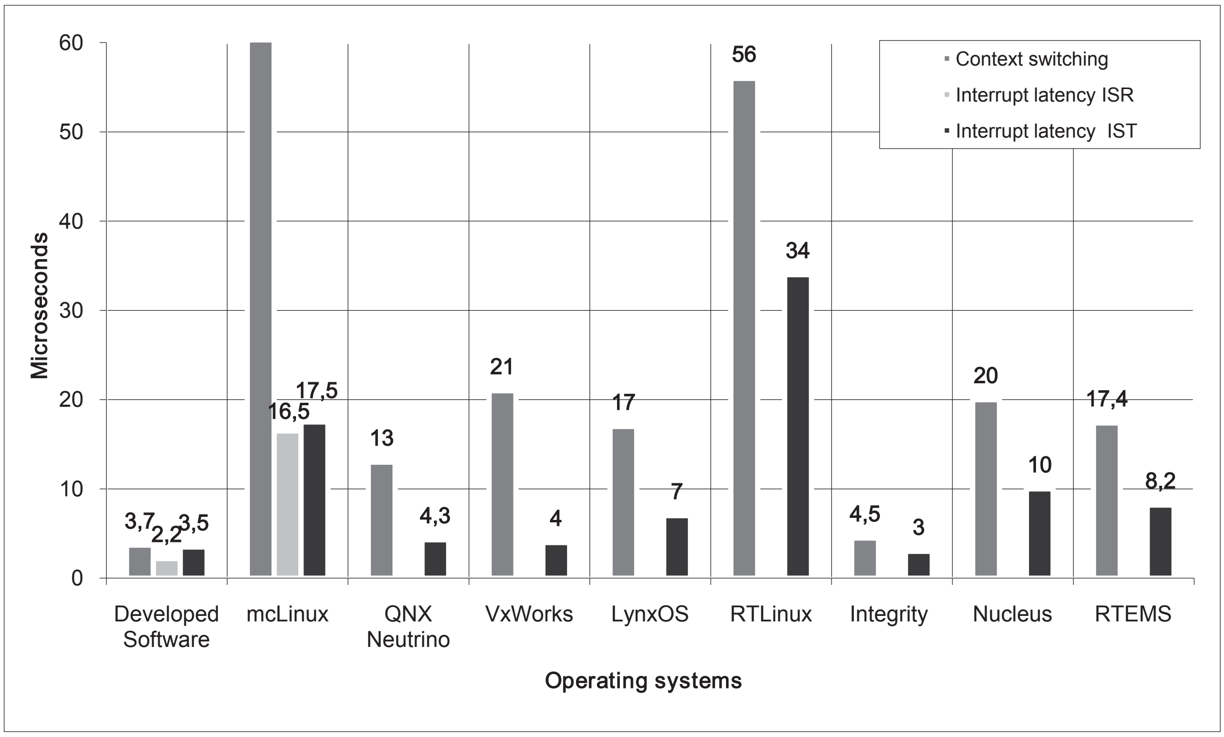

Figure 3. Time response characteristics

The stability of the time characteristics determines the predictability of the data processing time, the instant of issue of results, etc. QNX advances by this criterion VxWorks as on dispersion of characteristics in a series of the same tests, and with increasing load (thread switching time at magnification of number of threads 2-128 at QNX have grown only in 1,65 times whereas at VxWorks — in 2,24 times). LynxOS has the approximately same characteristics as VxWorks.

One more often used characteristic of embedded OS is required resources for system operation. The most often specified resources are minimum size of the kernel and required memory. On the in Figure 4 is shown chart comparing the minimum memory requirements of embedded operating systems. As you see, own developed software surpasses many of its competitors, the minimum configuration which occupies about 10 kilobytes of ROM and 800 bytes of RAM.

Presence of the free development environment mostly determines the quality and speed of development of new OS modules and applications for it. On this parameter own developed software is not far behind competitors such as LynxOS, VxWorks and QNX, which currently provide good quality tools, including integrated development environment application software.

On quality differentiation of access to critical resources of a computer system, such as memory and CPU time, the leaders are LynxOS and QNX, supporting both the process model and segmentation. The main task of the processes is the concurrent access to memory between different programs during execution time. Segmentation also provides separation of address spaces. VxWorks has a mechanism of segmentation, but does not support the process model. In general, it is permissible, since segmentation provides separation of address spaces. But as the segment will consist of threads with shared memory, it would complicate to organize parallel development and reduce the reliability of software.

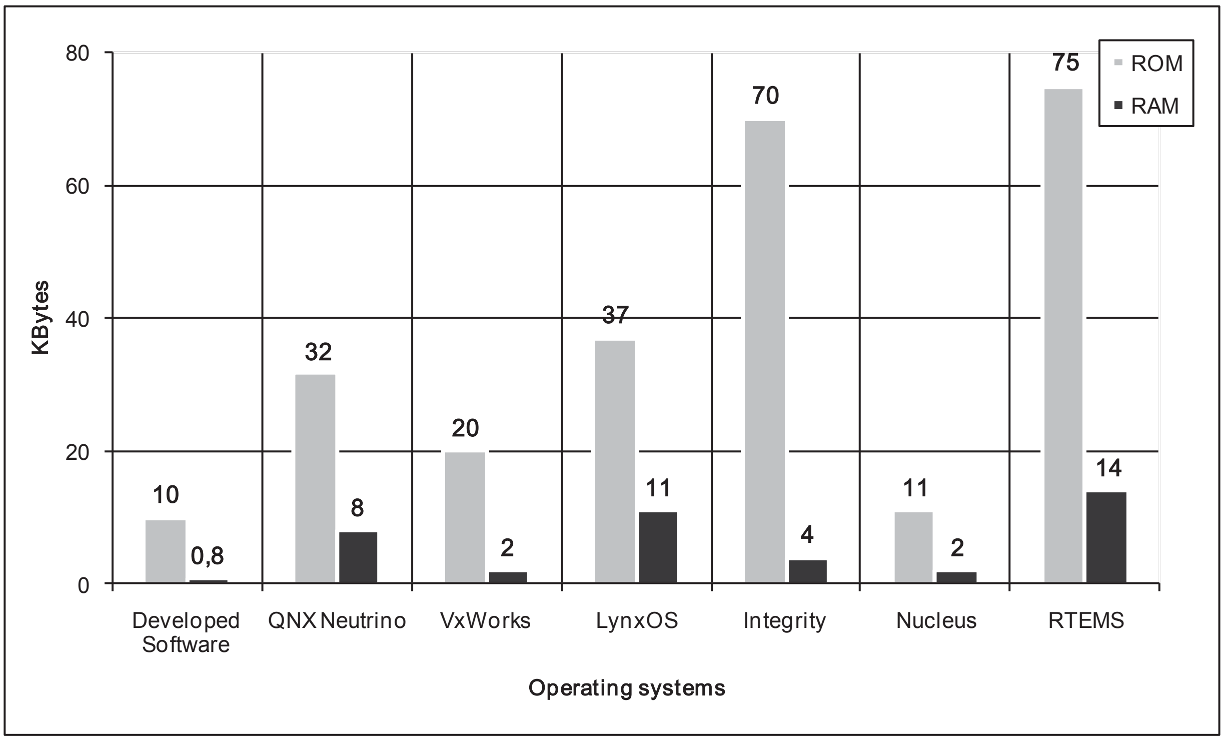

Figure 4. Minimum memory requirements

POSIX standard allows embedded OS with POSIX support easily transfer the programs from other POSIX-systems and draw in software development professionals who are familiar with these systems [1]. Compact and “fast” embedded OS can be developed according to the minimum profile PSE51 of POSIX 1003.13-2003 standard and it must supports preemptive multitasking, two algorithms for scheduling threads, mutexes, priority inheritance, timers, messaging and has the functions of thread management. Such developed embedded OS must satisfy the requirement on the implementation of at least 32 priority levels and implements a part of functions of the POSIX Threads [3].

IV. Comparing Results

According to the results of comparing characteristics of existing embedded operating systems that support the microprocessor architecture MIPS32, QNX Neutrino is the most advanced operating system from a technical viewpoint, having a good set of tools and relatively low price. Its main competitors are LynxOS and VxWorks. QNX microkernel architecture ensures high reliability and modularity of the developed systems. An additional advantage is the openness of the source codes. Nowadays, because of inconsistencies to many important standards, QNX is not so widely used in the industry as its competitors - LynxOS and VxWorks. By many characteristics own developed software is not far behind world market leaders in the field of embedded OS with support MIPS32 microprocessor architecture. Embedded OS developed according to the above methods and algorithms can be used as a "core" in embedded systems for various purposes: industrial automation, active networking and communications equipment, image processing systems, video, audio, and onboard systems.

REFERENCES

[1] Kevin M. Obeland, “POSIX in Real-Time”, *Embedded Systems Programming*, 2001.
[2] C. L. Liu, J. W.Layland, “Scheduling Algorithms for Multi-Programming in a Hard Real-Time Environment”, *Journal of the Association for Computing Machinery*, 1973. - №1.
[3] POSIX P1003.13-RT application environment profiles, IEEE P1003.13/D2.1 February 2003.