Real Time Kernel for Embedded Systems

Emil N. Dimitrov[1] Stanimir D. Mollov[2], Kristian Dilov[3]

Abstract – **The paper presents the problems connected with a real-time kernel for embedded systems. The system kernel building, the system process building and the interactions between the applied tasks and processing of external events have been discussed. The developed real time kernel MSPIX, which is intended for embedded systems, has been analyzed. The results of the analysis have given an opportunity to state that MSPIX possesses most of the contemporary real time-kernels features.**

Keywords – **real-time kernel, dispatcher, system process, applied task, descriptor.**

I. INTRODUCTION

The high requirements to contemporary control systems are the main precondition for their continuous improvement and modernization. To decrease the necessary time and resource expenses, the control systems are realized with a possibility of quick and easy configuration, reconfiguration and adjustment. These possibilities are supplied by software of the system. Its main purpose is to distribute the system expenses among the control processes. Such a system s is called Operating System (OS)[1].

Each applied system is designed on the base of OS and given user's tasks. On the base of the user's tasks the functional purpose of the system is formed. The time of changing the functional action of the system is reduced. Only the user's tasks are replaced, while the basic platform is unchanged.

II. BASIC CONCEPTS IN REAL-TIME SYSTEMS

The Real Time Operating System (RTOS) is a combination of the system software, which allows performing a large number of user's programs at one and the same time. The user's software can interact with external environment as well as exchange some data with another software.

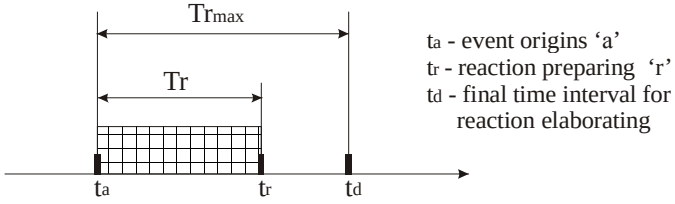


Fig. 1. Requirement for work in real time

[1]Emil N. Dimitrov is with the Faculty of Electronic Engineering, & Technologies, TU – Sofia, 1797, Sofia, Bulgaria, E-mail: edim@tu-sofia.bg

[2]Stanimir D. Mollov is with the Faculty of Electronic Engineering, & Technologies, TU – Sofia, 1797, Sofia, Bulgaria, E-mail: smollov@abv.bg

[3]Kristian Dilov is with the Faculty of Electronic Engineering, & Technologies, TU – Sofia, 1797, Sofia, Bulgaria

The term "real time" means that the system responds to each external event in a fixed time interval (Fig.1). The period T_r, which determines reaction R of the input influence A, is a variable because of asynchronous character and unpredictable influence on the computing process. The requirement of work in real time is reduced to the inequality:

$$t_r = (t_a + T_r) < t_d \quad (1)$$

where t_d is the final time interval for elaborating the reaction from Real Time System (RTS) [2]. The violation of this inequality is equal to the system failure.

The problems, which are solved during design of a real-time system, increase with adding the time co-ordinate. Under this condition it is necessary to create the real-time operating system as a multitask system. The multitask system consists of many asynchronous tasks and communication environment between them. Each task represents an active logical process, which is performing a kind of work within the system. In the common case each event is connected with a definite task.

The real time operating system provides a virtual process for each task. The virtual process performs the task in parallel to the others in the system. In the single processing systems this performance is a pseudo one – parallel or competitive. An actual parallelism can exist only between the tasks in multi processing or distributive real-time systems.

The hierarchical structure of the applied system (Fig. 2) can be divided into two components: a real-time operating system and applied tasks.

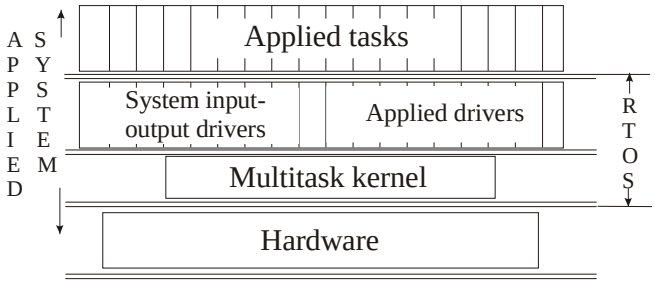


Fig. 2. Hierarchical structure of an applied system

The multitask kernel is located on the lowest level of the hierarchy. It implements all mechanisms, which support the multitask mode: task dispatching, the interaction between the tasks and events control. The kernel "services" are given to the upper levels in the form of system primitives and system macro. The system input-output drivers are an additional component and are used to control the standard peripheral devices. In contrast to the system drivers, the applied ones are additionally developed according to the used hardware.

The applied tasks are located on the highest level of the hierarchy. Their number, composition and character depend on the specificity of the applied system application.

III. SOFTWARE

A. System kernel

The main function of the kernel is to support the multitask mode. The code of the kernel is the most intensively used component in the system and in contrast to most of the drivers, it is a mandatory part of each applied system. Therefore the kernel must be effective enough with the requirements of a small memory size.

The purpose of the presented system kernel is to control the industrial controllers, which are built on the base of MSP 430 produced by Texas Instrument. The system kernel must create a possibility for competitive tasks performance in real time. The distribution of the processing time among the tasks is realized on a priority base by a dispatcher, which consists of two parts: a system clock dispatcher and an events dispatcher. The system clock dispatcher used Timer_A built in the microcontroller. It counts to a defined value and having reached it, generates a request for interrupt. This request is used for switching between tasks. After this counting starts again. The events dispatcher transforms the external influences in internal circuit events. The effectiveness of this transformation determines the field of the real-time system application. There are different methods to implement these reactions, but the most suitable one is that described above, which is used to interrupt from input port.

B. Applied tasks and their states

At the time of self-existing the processes in the real-time system have different states. The change of their state is done by the system kernel. This process is connected with the system losses – the processing time, the resource expenses. Because of this it is necessary to create exact rules for serving the tasks by the kernel.

At a given time each task in the applied system can be in one of these states:

- IDLE – the task uses only the memory area, in which its code is written as a resource;
- READY –the task has at its disposal the necessary conditions for its performance without the central processing unit;
- RUN – the task is being performed;
- BLOCK – the task is waiting for setting in some kind of event or resource discarding;
- WAIT – the task is waiting to pass a period of time;

At the beginning all tasks are in IDLE state. The procedure for establishment the task in READY state includes: creating the necessary data structures and stack segment organizing.

The tasks can be interpreted by the kernel as system tables – descriptors (fig.3). The manipulations, which the kernel can perform with system tables, are reduces to modify their fields. Each task has a unique *ID*, which is stored in the kernel work area. The kernel defines this *ID*, when the task is created. In fact this *ID* is the pointer to the task source in program memory. The current state of the task is written in field – *Type* and can be one of these enumerated above. The next field contains the task priority. The contemporary real-time system has 64 user's tasks. Because of this the maximum value, which can be written here is 64. The connection between the value and priority is straightforward (higher value – higher priority).

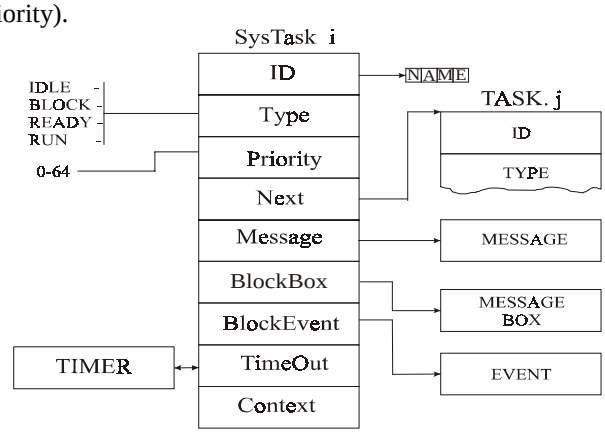


Fig. 3. Task descriptor

The arrangement of the tasks in the queue depends on the task priority – field *Next*. The beginning point of the queue contains the address pointer to the descriptor of the task of the highest priority. The fields *Message* and *BlockBox* in the task descriptor are used for communication between tasks, while the field *TimeOut* is used to definite the time interval in WAIT state. In the last field of the task descriptor named *Context* the work registers contents is written. These registers are: program counter, stack pointer, status register and accumulators.

C. System process

Before inserting a number of applied tasks, a system process is started in the system that in consequence originates them. This system process exists and is performed together with the applied task. Analogous to the applied tasks, the system process has a descriptor as well (Fig.4).

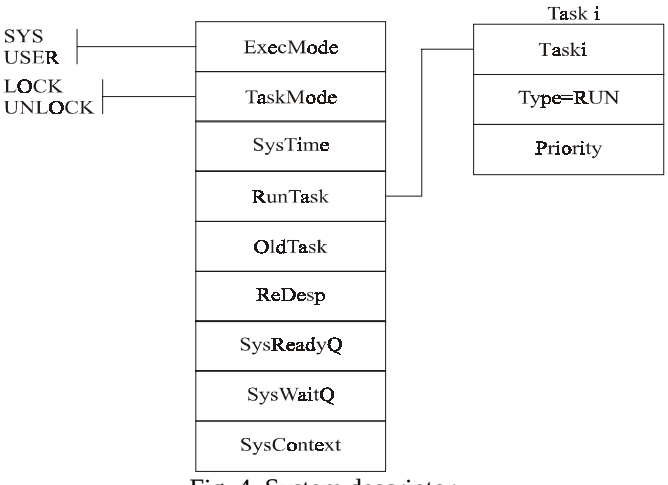


Fig. 4. System descriptor

The first field of the system descriptor indicates the current operating mode and can be:

- USER – the interrupts are enabled and there are premises for task dispatching;

- SYSTEM – the system manipulations are performed.

The field *TaskMode* defines the mode of the active task, which is usually UNLOCK. The task, which is in LOCKED mode, cannot be switched by another task and therefore it is working in a monopoly mode.

The system quantums are generated by Timer_A and heaping in the 32-bits counter –*SysTime* field. The system descriptor contains a pointer to the active task – *Runtask* field. It is possible that there is no active task at the moment (*RuntTask* = NULL). The pointer *OldTask* and *Redesp* are used for re-dispatching.

The system kernel keeps two system queues. The first one is the queue of ready tasks, which contains pointers to the tasks in READY state– *SysReady* field (fig.5). The second is the queue of holding tasks (Fig. 6). The arrangement of the tasks here occurs according to the contents of *TimeOut* field of their descriptors. The task with the shortest timeout stays at the beginning.

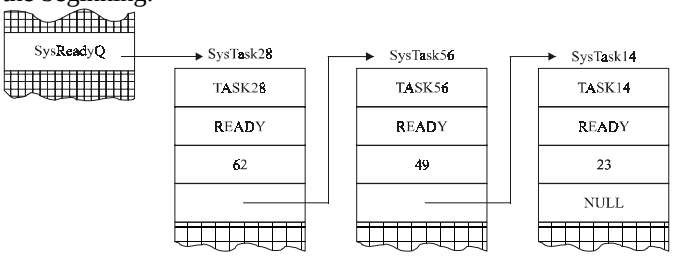


Fig.5. Ready task queue

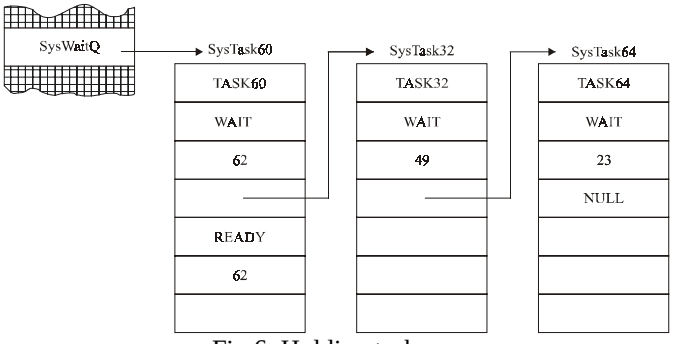


Fig.6. Holding task queue

The last field of the system descriptor contains pointer to system stack and status register.

D. Dispatchiring

The multitask kernel of the real time operating system supports a virtual processing unit for each task. Immediately after starting in the system exists only one virtual processing unit, which is intended for system process. When the applied task is creating the system process adds its virtual processing unit, which is executing in parallel with another tasks. In the single processing systems the processing unit is only the resource, which is distributing from the multitasking kernel. Therefore the dispatcher is a kernel component, which is distributing the time of processing unit between the ready for execution tasks.

There are two main strategies for system time distribution:

- Strategy without interrupts of the current executing task;
- Strategy with priority interrupts of the active task.

The strategy with priority interrupts gives privileges of the high priority tasks in processing time distribution. Because of this it is a more suitable strategy for realize the real time operating system.

It is possible several tasks to be with same priority. In this case they are switching alternate. The switching will continue while the change in the system occurs. The changes may be:

- An event, which is changing the state of the current task and removing it from list of the ready tasks;
- An event, which is setting the task with higher priority from the active task (fig. 7).

When the system quantum is time out, the dispatcher is checking whether there is a task with higher priority and if absence of such, the current task execution will continues in the next system quantum.

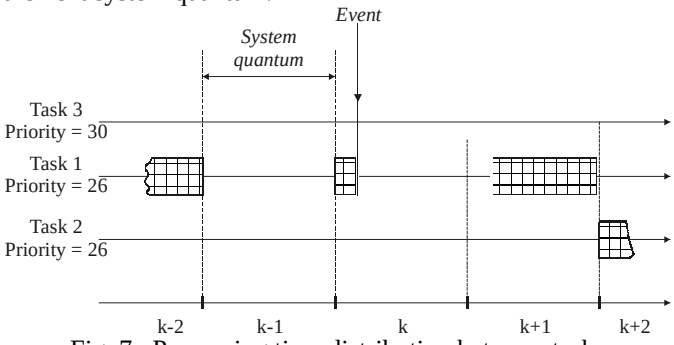


Fig. 7. Processing time distribution between tasks

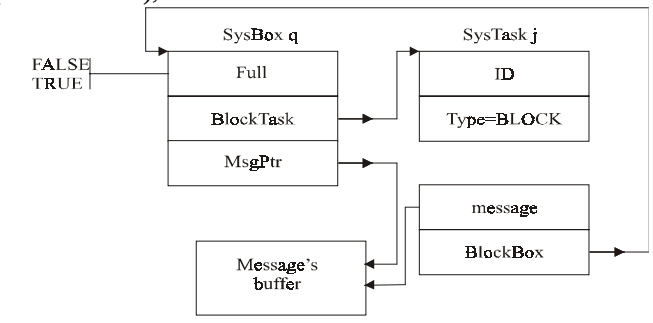
The system quantum is creating on the base of interrupts from the Timer A. During the interrupt service procedure the content of the current performed task is stored. Afterwards the processing unit registers are loaded with the contents of system virtual processing unit and the current mode is changed from USET to SYSTEM. The first task is loaded from the ready tasks queue. The content of the system process is stored and the registers of the physical processing unit are loaded with the contents of the first task from the list of ready tasks. After this the system mode is changed from SYSTEM to USER by modification of *ExecMode* field in the system descriptor.

E. Interaction between tasks

The interaction between the processes in the real-time operating system consists of data exchange between them. The data exchanged between the processes can be examined as a message of a defined format. The same idea is grounded on the base of the message box mechanism. The message boxes are used as a buffering message as well as for including the synchronizing methods between the processes: process transmitter and process receiver.

The message box descriptor is shown in Fig. 8. The flag FULL accepts FALSE if the message box does not contain any massage, and TRUE if the message is received by it. The *BlockTask* field contains the blocked task *ID* from the message box. This can be the task receiver, which is waiting for a message or the task transmitter, which is waiting for the message transmits. The message box buffer

is pointed by the pointer MsgPtr. When the box is empty (Full=FALSE), its contain is indefinite.



Full - flag full/empty Message box
BlockTask - blocked task identifier
SysBox - message box descriptor
SysTask - task descriptor

Fig.8 Message box descriptor

F. Registration of external events

One of the main functions of the multitask kernel is to transform the external influences into internal events. This transformation, which defines the area of the real-time system application, is performed by the embedded events mechanism (Fig.9). On the external event ei replays the system event Eventi, which is serving by Taski. It is possible that some events are served by one task, but each event can be served by one task.

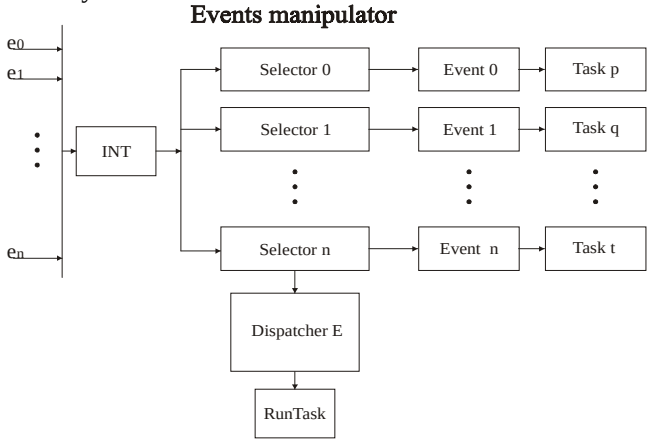


Fig. 9. Model of the event mechanism

When the event ei occurs, the hardware interrupt mechanism returns the control to events manipulator. The Selector identifies ei and establishes a connection with the system event Eventi. After this the system manipulator generates a request to the dispatcher. The events dispatcher distributes the processing unit time on the base of the strategy with priority interrupts: the blocked task on Eventi is activated if its priority is higher from the current performed task. In the opposite case, the task is included in the ready task queue.

The realization of the events mechanism depends on the architecture of MSP430 microcontrollers. To determinate the external influence two ports are used: Port 1 and Port 2 of the microcontroller. These ports have different interrupt vectors. The connection between the influence and the internal event is set by the system manipulator functions Selector0 – Selector.

IV. Real Time Operating System MSPIX

MSPIX is a priority multitask operating system, which is designed for use in embedded system built in on the base of MSP430 microcontrollers. Its parameters include: the necessary volume of data and software memory and some basic time parameters. The first parameter group is necessary for the microcontroller choice. The limitations here are a result of data memory size. It varies from 256 to 10240 bytes. It is recommended to use the microcontrollers with minimum 1024 bytes of data memory. The system expenses for a user's task are connected with creating its descriptor in the memory and with organizing its stack area. The task descriptor consists of 8 fields with a size of 16 bits (one word) and a field with twenty words for task content. The additional system expenses are created from the system descriptor (generally 34 words), message box descriptors (6 words) and event descriptors (2 words). Another basic feature of the real-time operating system is the time switching. These are quite important parameters characterizing the system. The switching between the tasks is done when the system process is activated. The system is in SYSTEM mode and the mask-like interrupts are not allowed. At this moment a risk of event service delaying can occur. The time parameters are shown in Table 1.

Table 1

Expenses	Cycles	In seconds
Maximum switching time between the tasks	496	66.10^{-6} sec
Minimal switching time between the tasks	344	43.10^{-6}sec

V. Conclusions

The developed real-time operating system possesses most of the contemporary real-time operating system features.To analysis the main system parameters, it is developed a system on base of MSP430F149 microcontroller. In this system the maximum switching time between the tasks is **66us**, and the system quantum is 66ms. The developed system is characterized with low processing time expenses. The results of the developed system are near to the contemporary real-time operating systems.

V. References

[1] Henzinger T.The Embedded Machine:Predictable, Portable Real-Time Code, EECS, University of California, Berkeley.

[2] Дилов К. Дипломна работа на тема: Операционна система за реално време за фамилия микроконтролери MSP430, София 2004.

[3] MSP430X1XX Family User's Guide - Texas Instruments.

[4] Single-chip Microcontroller real-time operating system – Digital Cellular Magazine 2002.

[5] J.K Stankovic, J. K. Ramamritham. The design of the spring kernel. Real-time systems symposium, San Jose, 1987.