Real-time control implementation for MS Windows 2000/XP operating systems

FOJÍK DAVID, BABIUCH MAREK
Department of Control Systems & Instrumentation 352
VSB - Technical University of Ostrava
av. 17. listopadu 15, CZ 708 33 OSTRAVA - PORUBA
CZECH REPUBLIC

Abstract: - The article describes the add-on implementation of MS Windows NT operational system for real-time control. It is based on a possibility of executing time critical operation at the lower layers of an operational system, in particular at the interrupt handling. In this way timers of added devices (conventional multifunction measuring cards) are used for precise sample period generation within tens of microseconds order as IRQ request form.

Key-Words: - Real-time, MS Windows, Automatic control, Data acquisition card, Interrupt, IRQ

1 Introduction

Architectures of PC type computers are the subject of the permanent interest of control systems designers. Their relatively low price, permanent increase of power, wide support of modern software and hardware components and variety of producers are strong arguments for their use. Furthermore, an important fact is that they are at the present time, essential and sometimes the only one platform for the entrepreneurial information systems. Applications for PC therefore offer the possibility how to easily build quite uniform company system. This will significantly reduce investment into them.

In order to satisfy this requirement it is also necessary to unify used operation systems. The operation systems of the MS Windows NT type obviously dominate the PC platform (this summary name covers, to make it clearer, the operating systems of MS Windows NT, MS Windows 2000 and MS Windows XP). In this connection, the legitimate requirement appears for using these named operating systems also for the operation of applications working in real time.

The producer himself acknowledges this fact, since he offers the specialized versions of MS Windows NT Embedded and MS Windows XP Embedded operating systems, and what is more, he takes part in the development of real time RTX complement of a high quality (*Real Time eXtension*; by VenturCom Company). In certain cases this complement doesn't have to entirely suit the user. Not because it would not enable the implementation of some tasks, but rather the opposite it can require simpler, cheaper and already specialized solution, which will be immediately ready to use without further need for understanding new environment and creating new algorithms. This contribution is describing one of such specialized software solution: The complement for the area of a real time control.

2 Real-Time and MS Windows NT architecture

It is commonly known that the Windows NT operating systems are not designed for real time execution. This has been the topic of many articles before, which give detailed explanations for all the reasons. We will not discuss here again, which criteria the Windows NT system does not satisfy and why it cannot be considered as the real time system, so we will only focus on those aspects which essentially complicate their use of specialized applications for control and monitoring in real time. The systems requirements for these applications are at the minimum the following:

- Immediate access to hardware – I/O unit,
- Precise, fast and always reliable timer, which ensures repeatable awakening of a control task within a given moment (control sampling period),
- Uninterrupted operation within the time period, consisting of acquisition of the controlled variable, the computing of the control algorithm and the execution of the actuating variable.

  In order to understand how the Windows NT operation systems satisfy or fail the mentioned requirements it is necessary to briefly mention their architecture.

Firstly, it is necessary to say that the system distinguishes two working modes: the user and the kernel mode. All the running applications (processes) without any exceptions work in the user mode, from which they cannot directly access the hardware. Processes depend, in this way, on drivers for those devices, with which they communicate via Win32 API subsystem services.

Each process (running application) owns at least one computing thread, which executes process code. Each thread of the all processes is alternatively processed by the processor. The scheduler decides which thread

should run, according to current thread's needs and assigned priorities. For example, the thread with the highest priority can be put asleep for a certain time or it can wait till the source detaches, which will release the run of threads with the lower priority. Time and synchronisation functions of Win32 API (in short timers) work on the similar principle. The thread simply goes to sleep after a certain instruction was executed, and waits until the timer wakes it up again.

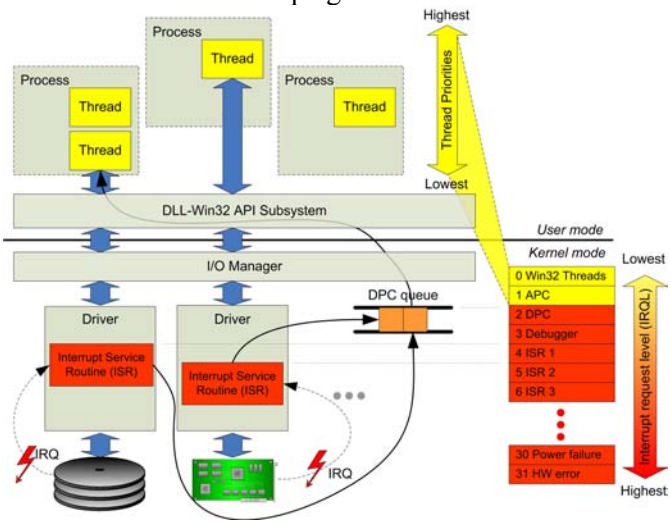

Fig. 1 MS Windows NT architecture - all priority levels

For these purposes the system offers more than just few tools, which can be theoretically set with accuracy up to one millisecond. In practice, such accuracy can only be reached under certain conditions with a multimedia timer. Nevertheless, even this timer cannot be used for the real time operation. The reason lies in the scheduler itself, which in addition depends on other priority settings, so called the IRQ (Interrupt ReQuest) level. The system distinguishes 32 of such levels, where the highest priority has the IRQL 31 level and the lowest IRQL 0 level. And the scheduler (as well as all of the threads) runs just exactly at the lowest IRQL 0 or IRQL 1 levels by their priority. All interrupt handling services and tasks from DPC (Deferred Procedure Call) queue run at higher levels, which calls interrupt handling again. These tasks are not mostly critical, however they are very often time consuming.

3 Functional principle of the complement design

The above mentioned imperfections can be avoided with postponing timely critical operations into the lower levels of an operating system, more precisely into the interrupt handling of hardware timer. Just this option is the base for this presented design, which utilises abilities of the most multifunctional data acquisition cards available at the present time. It is the interruption generated usually as a consequence of the end of one or a set of samples measurements. As a reaction to this interruption, the handling of specially designed driver is initiated, the entire code of the control algorithm will be executed then. The user application serves the role of only some intermediation; with which the user selects a type and parameters of the controller, or visualises or processes in some other method data with information about the course of control. This principle is explained the best in the following text and Fig. 2:

1. the user application transfers, at first, to the driver according to user choice data about a controller type, including all input parameters used, and so on,
2. then the driver introduces the certain interruption handling, sets the measurement multifunction card's circuits and initiates the timer so that the required sampling period is ensured,
3. since this time, accurately and absolutely independently on the computer state, the timers of a data acquisition card runs repeatable the A/D conversion, so the card generates an interrupt request,
4. with this request the hardware finds out together with the operating system the priority, and if it is evaluated as the highest, then it begins immediately its handling,

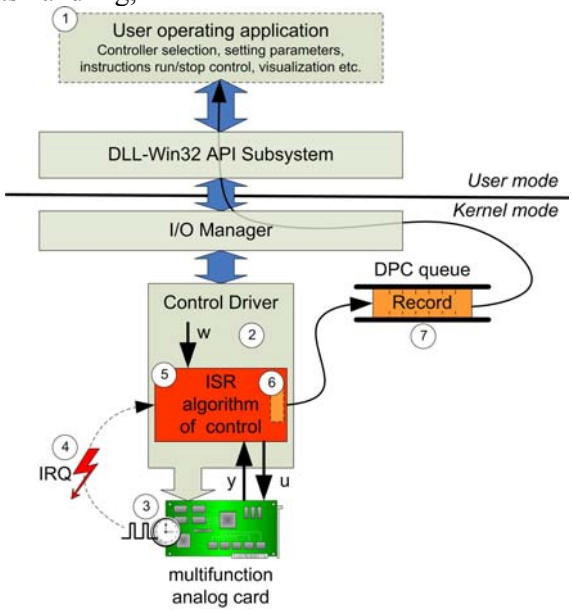

Fig.2 Principle of a complement's activity

5. interrupt handling reads the already measured input value, executes the certain computation and immediately sets the output,
6. if the user application requested for continuous information earlier, this routine, will before its end write the variables' states into a cache memory.

After the writing into a cache memory the application hands over its content through this task into DPC queue back to the user application, which will process the data according to the user. From the point of view of the real

time needs this design brings many advantages. Let's mention just the most interesting ones:

- Since the timing is generated by independent hardware counters, it is possible to select sampling period independently on the system usually with the accuracy of one microsecond. In addition with most of the cards, the counters will initiate A/D conversion themselves and only after they generate the interruption; in this way, the sampling is absolutely precise and at the same time, the time delay does not occur while waiting for conversion's end.
- Since all drivers' activities are performed in the kernel mode, it is possible to access directly and without any limitations to computer's hardware, it means to the registers and a memory of the card.
- The interrupt handling executes at the high IRQ level so that the system tasks hand over data about the course of control including the thread activities, which are processing them, they cannot influence the control itself.

As it happens, the advantages on one hand are accompanied by the disadvantages on the other hand. This principle also brings several complications, which must be solved or significantly limited. The fundamental complications of the described design are:

- The realisation is very program demanding. Design of drivers requires experienced programmer with knowledge of C language, Windows NT architecture and I/O system function principles. The necessary compilers and quality documentation with examples can be acquired in DDK (Device Driver Kit) set at the Microsoft web server. Driver debugging requires in addition another computer with a special version of operating system installed (Check Build). (all detailes can be found in DDK.)
- The drivers do not have support for floating point operations, which is essential for control algorithm design.
- The algorithm on its own is executed by driver therefore any adjustment of existing or added new algorithm requires an intervention into this driver.
- The driver is dependent on a concrete data acquisition card type, so that other types require a new design of a driver.
- Each type of a controller can have different setup parameters, so any change or addition to the algorithm requires similar intervention into the communication protocol and the user application.
- Although sampling with hardware counters and interruption generation are absolutely accurate, however they do not guaranty immediate initiation of handling. As it was previously mentioned, individual handlings are processed according to IRQ levels, so that the algorithm execution itself can be delayed while the handling is performed with a higher priority. At the same time, trouble occurs with computing the actuator's action based on a relatively outdated value. It is obvious that the problem is the more significant the lower priority the handling has and the more the interruptions occur of a higher priority.

4 Implementation and structure of a complement

In the previous chapter, the functional principle of the software solution together with its problems was shown. The following text is focused on the description of the implemented complement, which functionally correspond to this principle. The complement is not a final solution, but it is only a specialised „semi-product" offering a set of mutually cooperating libraries and modules, which enable the easy building final solution. The complement solves many of the mentioned problems. Other problems (for example the problem with various time responses to incurring interruption) are fully solved at the levels of individual modules and libraries.

4.1 Minimisation of the variance of time responses to incurred interruption

From the point of view of the brief description it can be wrongfully concluded, that the entire problem can be solved within the increase of the IRQ level. Unfortunately, this is not the case. It is all much more difficult and depends on many factors, such as on the kernel implemented mechanism of interrupt handling, which is not unified and is significantly influenced by a computer hardware (namely by a interrupt controller) and the version of an operating system (it works differently in Windows NT, in Windows 2000 or Windows XP).

Two types of controllers can be found at the PC platform. The older one, usually marked as PIC (Programmable Interrupt Controller) is based on cascade connected circuits 8259A, which were used with Intel 8080 processor already. In spite of its age it was exclusively used in all single processor systems until the Pentium 4 processor release. The newer controller is the APIC (Advanced Programmable Interrupt Controller), which was developed especially for the need of multi-processor systems.

From our point of view it is interesting to see that, when the older controller is used in the computer, the IRQ level of the corresponding handling is unambiguously given by the interrupt channel. It is consequently always clear at which level the one or the other request will be processed in a standard way.

Table 1: IRQ levels and the IRQ request relationship in systems with PIC circuits

IRQL	IRQ	Description
31	-	Machine checks or bus errors
30	-	Power failure notification
29	-	Interprocessor request
28	0	Timer
27	-	Profiles
26	1	Keyboard
25	2	INT from slave PIC
24	3	COM2/COM4
23	4	COM1/COM3
22	5	Reserve /Sound card
21	6	Floppy disk
20	7	Parallel port
19	8	Real-time clock
18	9; 2	IRQ2 or IRQ9
17	10	Reserve
16	11	Reserve
15	12	PS/2 - computer mouse
14	13	Mathematic coprocessor
13	14	Hard Disc
12	15	Reserve
4-11	-	No use
3	-	Debugger execution (SW interrupt)
2	-	DPC (SW interrupt)
1	-	APC (SW interrupt)
0	-	Execution Win32 API threads

When the newer APIC is used, this relation does not exist. Each system installation can have different levels adjoined. In other words, the same version of an operating system with the same hardware can have adjoined a different IRQ level for the same channel. So far, the relation of the complement with a standard interruption controller, which can be very often used with the newest systems through BIOS, is satisfactorily being solved. With specific algorithms the change of behaviour was reached so that handling of a defined request is almost in all cases preferentially performed. The exception is when handling of another request has been already initiated with initially higher priority. But even after this it is in the imaginary queue of awaiting requests in the first position. The resulting effect can be shown the best in Fig. 3. It can be seen here the relation of the time response variance on the initiated interruption cased by absolutely the same conditions at the same system, however with the primary interrupt handling first, and then with the adjusted one.

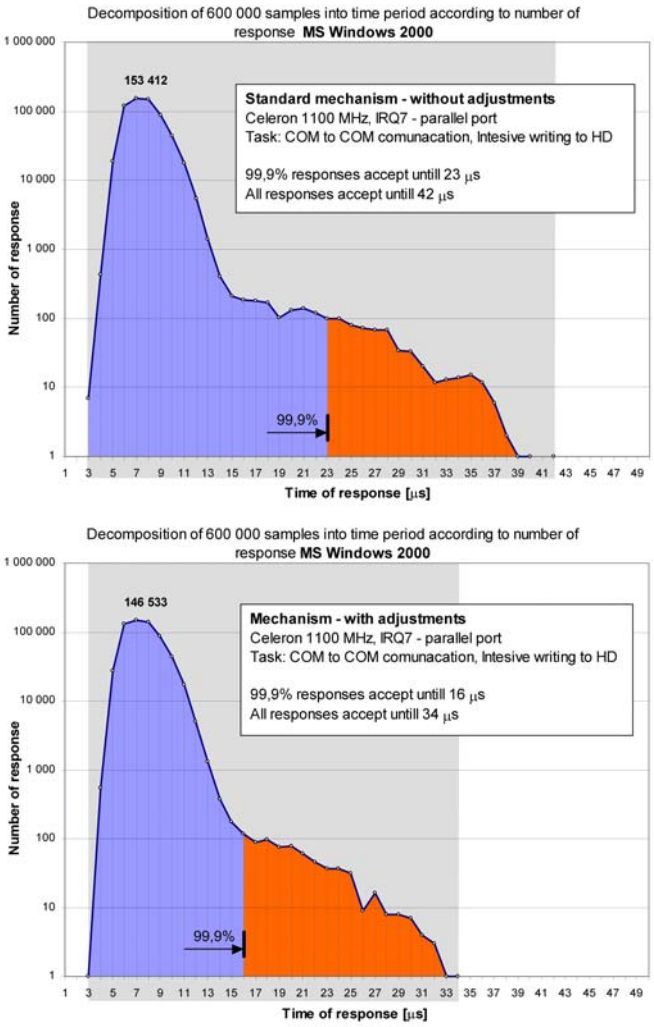

Fig. 3 The difference in time response variance between the standard and adjusted mechanisms

4.2 The complement architecture

The whole complement consists of three mutually cooperating parts:

1. a low level driver,
2. levels simplifying communication between driver and applications,
3. user control application.

These parts together create an easily implemented complement, which enables immediate realisation of control for SISO (single input single output) circuits. For MIMO (multi input multi output) circuits it is necessary to complete the driver with a corresponding algorithm. The driver itself is commonly constructed, which means that it requires adjustment for a concrete type of multifunctional card.

4.2.1 Common low-level driver

The most important part is commonly designed low level driver, whose architecture (Fig. 4) enables easy adjusting to an optional type of a multifunctional data acquisition card, which however satisfies necessary requirements for timer existence and ability of interrupting generation. The driver itself consists of three layers:

- Interface layer,
- Logic – functional layer,
- Physical layer.

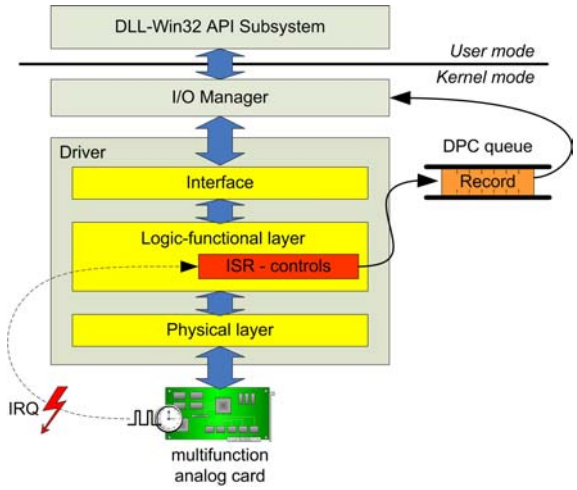

Fig. 4 Driver structure

Kernel is created with a logic – functional layer, which ensures all functions (including control, loading and driver installation, etc.) and saves all settings. This layer does not accesses data acquisition means nor communicates with Win32 API applications. These activities are ensured by two neighbouring layers, which it closely cooperates with. The physical layer is used for the direct access to hardware, which depends, as the only one, on a concrete data acquisition card. The lowest layer of the interface ensures communication with Win32 API environment. This one is created in a way that it is not dependent on a concrete type of a multifunctional card (it covers, therefore, all accessible types) and it enables easy extending of a functional layer with new types of tasks, as well.. Thanks to this concept it is very easy to transfer solution on another type of a multifunction data acquisition card – basically it is necessary to modify only the function frames of a physical layer.

Driver architecture also keeps in mind the extension by other types of controllers. Internally, the logic – functional layer is divided in a fixed part, which does not change, and a part, in which control algorithms are implemented. This part is divided not only logically, but physically as well; therefore new algorithms can be easily implemented. Common types of controllers are implemented in the driver as the standard utility. For example, PID controller is available here in its common positioning and an incremental version with the option for adding filter either separately with a derivative part or a controlled variable.

The functional library, which implements basic arithmetic operations with floating numbers, is created especially for this purpose to ensure design of new algorithms without problems. These algorithms can be created separately from the driver in a prepared application for Windows, which makes the design significantly easier by simulating the driver activities, and enables transferring these algorithms after they were debugged into a driver.

4.2.2 Communication layers

Although the driver is able to control applications on its own, it will not manage without communication with Win32 API applications (Fig. 5). The driver needs to know, at least, the type and parameters of a controller, used inputs and their voltage range and needs to get a command for starting control. Further more, the application may require information about a course of control, which is consequently processed, etc. It is obvious, that communication is very important.

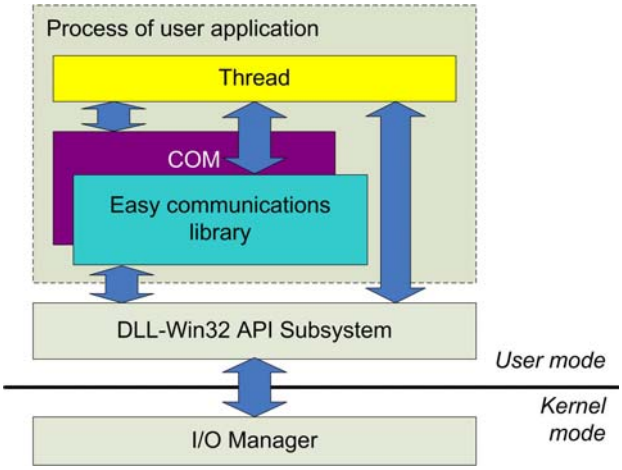

Fig. 5 Communication with a driver

The communication is cared about at the level of a driver by an interface layer. Any application has the access to this layer through Win32 API subsystem. The direct communication with an interface layer is not, however, easy. There is a great demand on all programmers to know much about not only principles of this interface, but common knowledge about communication between applications and a driver. That is why a library of functions for easy access was created to make their work easier. In its own principle, this library overlaps the common Win32 API subsystem function with its own functions, which were especially developed for data exchange with a driver interface.

But even this library is not easily being used, especially when an application is created in other programming languages than C or C++. That is why a COM component was created, which "puts" the entire communication into an object-oriented structure. A component is again designed with regards to other extensions of a driver or another data acquisition card. Thanks to this component it is possible to create applications in any programming language, for example preferred MS Visual Basic and similar.

It is possible to integrate the driver through these mentioned tools into any application, while the programmer can choose a method of communication, which suits him the best. He can then decide according to current options or needs, or he can choose a combination of those methods.

4.2.3 User application

The last part of a design is a user operating application, which intermediates all functions of a complement in a user-friendly environment. Shortly, this application enables selecting and setting all supported types of controllers, monitoring and visualising a course of control and exporting data recorded into text files, or MS Excel files respectively. An application is not, again, bound to a concrete type of a driver, therefore it cooperates with any data acquisition card, or with any compatible driver.

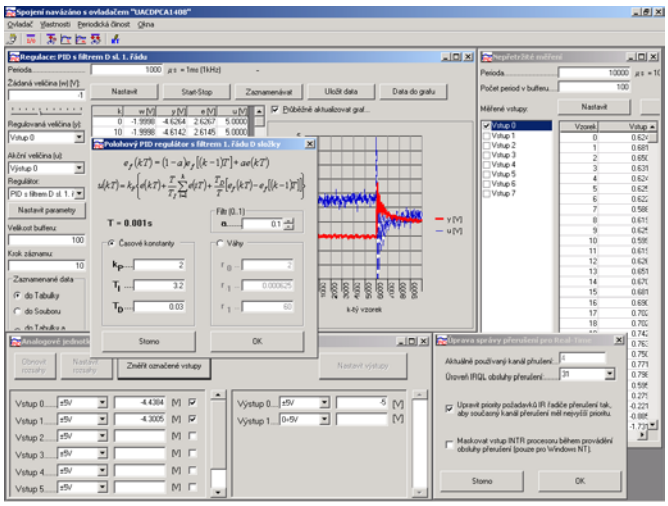

Fig.5 User operating application

5 Conclusion

The complement is with its specialisation and, at the same time, with an adjustable architecture at the borderline of a development mean and the final solution. The end users will be rather interested in an already bound complement with the concrete multifunctional card. A common driver solution is, for a change, interesting for the data acquisition card producers.

At the present time the driver is implemented within two data acquisition multifunction cards, PCA 1408 and PCA 1208. In both of the cards all functions and properties were verified as well, the algorithm functionalities were verified on the control model of electromagnetic levitation. To make it more unbiased the driver was tested on several system settings with different power, with a range of processors from Pentium 90 MHz up to Pentium 4 with 2 GHz frequency. The system was always able to safely ensure the control with sampling frequencies at least of 10 kHz. The system was intently loaded during the tests by selected tasks (for example copying CD-ROM content to a hard disk drive, network communication and so on.). These tasks, however, fundamentally never influenced the sampling periods, or the control itself.

This solution „does not catapult" the operating systems of Windows NT architecture into the system category of the real time operation, however, it enables easy implementation of the control of systems demanding promptness and runtime execution. It is, in this way, an interesting alternative for the control area in laboratories and in industry, where architecture of MS Windows NT is requested together with high flexibility and prompt and easy implementation.

References:

[1] FOJTÍK, D. *System Modules Implementation for Process Control Support at MS Windows NT Operating System.* Ostrava: Department of Control Systems & Instrumentation 352, VŠB-TU Ostrava, 2004. 188 pages, Dissertation thesis.

[2] MICROSOFT *Microsoft Windows 2000 DDK*. Redmond: Microsoft Corporation Redmond USA, June, 2000.

[3] LANDRYOVÁ, L. *SCADA Applications based on .NET Architecture*. In 5th International Carpathian Control Conference. Zakopane, Poland : AGH-UST Krakow, 25. – 28. 5. 2004, pp. 313-318. ISBN 83-89772-00-0.

[4] ŠKUTA, J. *Web Cam Using for Real-time Tasks Monitoring in Control Web 2000 System*. In Proceedings of 3rd International Carpathian Control Conference. Ostrava : VŠB-TU Ostrava, 27. - 30. 5. 2002, s. 463-467. ISBN 80-248-0089-6.

[5] KULHÁNEK, J. *The Speed of Component-Based Application in .NET Platform.* In 5th International Carpathian Control Conference. Zakopane, Poland : AGH-UST Krakow, 25. - 28. 5. 2004, pp. 843-848. ISBN 83-89772-00-0.

[6] WAGNEROVA, R. Robust Control Design for Technological Processes. In *5th International Carpathian Control Conference*. Zakopane, Poland : AGH-UST Krakow, 25. – 28. 5. 2004, pp. 289-294. ISBN 83-89772-00-0.