BULETINUL Universităţii Petrol – Gaze din Ploieşti	Vol. LXI No. 3/2009	293 - 300	Seria Tehnică

Regarding Capabilities of Real –Time Programming Platforms CTask, Java and QNX

Eduard Valentin Niţă

BRD – Groupe Société Générale, Pantelimon 258 Ave., Bucharest
email: eduard.nita@brd.ro

Abstract

This paper proposes to evaluate capabilities of real – time programming platforms CTask, QNX and Java based on a case study. The case study is represented by the development of an application for real-time control of a flow through a pipeline. The control algorithm used in this paper is the PI algorithm. This paper ignores the type of control devices (transducers, actuators, acquisition boards), focusing only on real – time programming and on the development of the control application.

Key words: *real – time, task, capabilities, CTask, QNX, Java.*

Introduction

Real-time information processing (critical time processing) is an extremely important domain only that near nobody don't give enough importance to this area. The way in which a car is driven is a real time problem. Alike, is a web site that guarantees answers to a certain query in maximum two seconds, for example, or a text editor which must respond to every keystroke in matter of hundreds of milliseconds in order to avoid discomfort to users.

In general, a software developer is very concerned about the modules in the application and very often he does not regard the global performance issue. In most cases the software developers focuses a lot on the user interface and on the speed performance of that application and they ignore completely organizing the execution.

Although, at this time there are many computers with extraordinary performances, practice has shown that the developer must write a program of which he will be sure that it shall end its execution until a period of time elapses. Algorithm implementation using real-time mechanisms does not produce a more rapidly execution of applications, as it may be thought , but it drives to a sequentially execution, thus that it can be said that to a certain moment of time one single and known action is in progress.

Theoretical considerations

Real-time applications are built on the strength of a tasks system, tasks that represents fundamental entities; and they are defined to do only one specific task of a certain operation.

Any task needs resources in order to be launch in execution. By resource we understand any component that a task needs in order to be run.

By consistency, the resources can be material, hardware and logical or software, and by the level of accessibility the resources can be local (they can be accessed by a single task) or shared (they can be accessed by more than one task).

A shared resource which admits at a certain moment of time only one session to be run it is called critical resource.

A shared resource which admits at a certain moment of time that upon it to be run more than one session, in other words n sessions, it is called shared resource with n access points.

A shared resource which admits at a certain moment of time that upon it to be run an undefined number of sessions it is called reentrant resource.

The section in which a task accesses a critical resource it is called critical section.

Real-time multitasking kernels are due to offer the methods which must permit the denial of simultaneously access of more than one task in their critical sections regarding the same critical resource.

The exclusion by one task, which it finds out in a critical section regarding a shared resource, of the possibility of other tasks to enter in their own critical sections regarding the same shared resource it is called mutual exclusion.

When a developer implements by software a mutual exclusion he must respect Tanennbaum's recommendations:

- ✓ one critical section can be run at a certain moment of time by a single task;
- ✓ it can not be said anything about speed or number of tasks which are running at a certain moment of time;
- ✓ any task has the right to enter in his own critical section;
- ✓ the tasks can not be blocked as long as they are in their own critical section.

Having tasks put in a certain temporal relation one with the others or in certain relations with external events it is called synchronization.

Implementing synchronization presupposes that the kernel must offer mechanisms that ensure [2]:

- ➢ activation or deactivation of one task;
- ➢ blocking current task or of another task;
- ➢ unblocking of one or more tasks.

Synchronization can be realized using external events or by time. For synchronization with events, the kernel must offer two types of functions [2]:

- ➢ a *WAIT* function which enforces the user task to pass into the state of blocked until a certain event is produced;
- ➢ a *SIGNAL* function which points out to the producer task the awaited event and which determines the transition of task from blocked state to ready state.

The methods an the mechanisms used for synchronization implementation are distinguished from one to another under many aspects like [2]:

- ✓ by the nature of synchronization (synchronization between task and with time);
- ✓ by the moment of synchronization (synchronization between two tasks: with the starting point of a task, with the ending point of a task or with an intermediary point);

- ✓ by the implementation of synchronization (by programming language facilitations or by some command language facilitations, and by some monitor facilitations).

Task – time relation can be synthesized by restriction. Operation *A* can not be run before Δt period of time expires.

Regarding synchronization with a time condition, in case that a task runs in an infinitely loop, may exist two situations like [2]:

- ✓ the task awaits for Δt period of time to expire, and after that it restarts again;
- ✓ the task is run in the Δt period of time.

From user's view, the task is run in the Δt period of time.

If the task is being run in the Δt period of time, then it can be two types of basic functions:

- ✓ one basic function that uses the name of the task and the Δt period of time, with the possibility of calling it from any other task;
- ✓ one basic function which has as preset parameter the Δt period of time and an event variable.

When the second basic function is called by the running task, the course of action is the following: the value *FALSE* is attributed to the event variable, then starts the time counting , and then when the period of time expires the *TRUE* value will be attributed to the event variable.

Inside of a task system there are disjunctive tasks and interactive tasks. Real time operating kernels must offer the means for information exchange without conflicts [2].

The mechanism through which communication between tasks is realized are: pipeline communication, message and mail communication and communication through general semaphores.

Real – Time Flow Control Application Using CTask Kernel

In the following part of this paper we will consider a flow process for which we desire to develop a real time control application. We will not consider in this paper the drivers and hardware resources needed for interfacing computer with the actuators and the transducers. Also, for this paper we will consider that the algorithm for control is the PI algorithm.

In the following we will discuss about the developing of an application for real-time control of a flow through a pipeline using the real time kernel CTask.

The CTask kernel was designed and developed by Thomas Wagner, and it is structured as a collection of functions linked in libraries. This kernel gives to the C programming language quasi-parallel processing capabilities [3]. CTask is running under MS-DOS and in combination with Microsoft Windows 98 or Me, as primary operating system, the system may become unstable. In combination with Microsoft Windows 2000 or XP, because these two operating systems does not have a native DOS platform, the MS-DOS must be launched through an emulation program and so the memory consumption is very high.

In CTask, a task is defined as a "far" C function. There can be defined as many tasks as the programmer needs, so there is no limit regarding the number of tasks in the system. A task function is different from a classic function of structured programming, and so a task function is never called directly. Usually a task function is specified in the call routine of the task creation and it is launched in execution through the task start routine. While running, the task shares the time processor with all other active tasks until is killed or finished by the task ending routine[3].

The application was developed on a four task system structure: a task which has as mean the flow control, based on a PI algorithm. The control is made, by the algorithm, based on a value acquisitioned from the process (flow of a fluid), and as a result the task will request the actuator to take an action based on the value generated by the control task. This task will run on certain periods of time, periods given by the clock of the system. When running, this task will request to the other tasks of the system to perform actions, and in those periods this task will be in a blocked state. In this manner, we accomplish a well determined sequential running, the mutual exclusion is realized implicitly (there are not used conditional variables or resources), and a planning on condition of time because the tasks are run each time on a constant period of time. This period may be inflicted by the developer depending on the application requirements and on the dynamics of controlled process.

The second task of the system is responsible with acquisitioning the value from process needed by the control task and he must send that value to that task. This task will run only when the control task makes a pointedly request for this and after that it blocks. The request is made by the control task which sends a signal for the acquisition task. This signal will be owned by the acquisition task until he runs completely, meaning that it will send the acquisition value to the control task. After that, the acquisition task will erase from system the signal generated by the control task and it will enter into a blocked state until the signal is generated again.

Third task of the system is the task responsible with the generation process and which has as main objective control value transmission to the actuator. As shown previously, this task will run only when the control task makes a pointedly request for this. After the request has been made, a signal has been generated; the control task will pass into the blocked state until the signal is erased from system. When the signal is generated, the generation task will run and it will transmit to the actuator the corresponding value. After transmission, this task will erase from system the signal which activated it and will pass into the blocked state.

The fourth task of the system is the graphic user interface task. This one has as main objective the on-screen display of the three characteristic values of the control system: the value of reference, the value of control and the value of controlled measure. Because of the quasi-parallel running of CTask and the adding of developing the application on a system which has the clock frequency very high, this task can be implemented to run on undetermined periods of time, without the danger of blocking occurrences or delays in system. The problems may occur, regarding the graphic user interface task, during running below old generation of computers (we refer especially to computers with microprocessors from 8086 generation). These problems may be avoided because the application may run without problems below MS-DOS operating system and the installation of a more evaluated operating system is not necessary. From this we can conclude that the microprocessor will not be busy with other threads or other services of the operating system, but only with the tasks of the application.

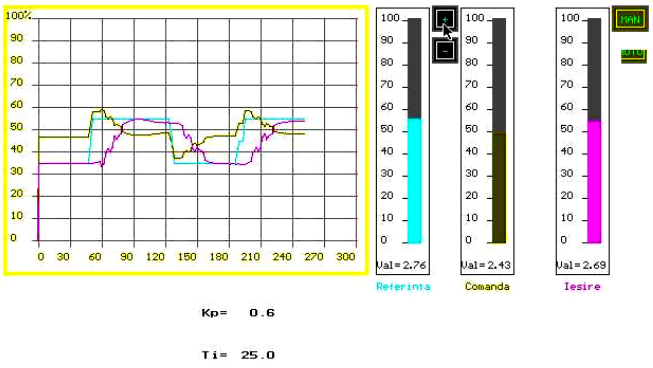

Fig. 1. Real – Time Flow Control Application using CTask Kernel – while running.

Therefore, as per the structure presented until now, the control task carries out the role of planner. It will made explicit requests for running all the tasks that it will coordinate [1].

In the system, there is the fifth task with special properties. This is the main task, implemented in the main function and that it has as main objective variable initialization, link with the libraries and the header files

needed for a good running of the application. Also, one of its objectives is making, initialization and starting of the tasks built around it, the explicit call of CTask kernel planner and the running of the tasks. At the end of the application it will kill all the tasks in the system and it will free all the resources used by the application. Still, this task is responsible with the initialization of the on-screen display and the objects that will form the graphic user interface.

Real – Time Flow Control Application Using Programming Language Java

Until now we have discussed about developing a control application using real time kernel CTask. In the following part of this paper, we will discuss about developing the same application using the programming language Java.

The application was developed using the integrated development environment NetBeans, a free tool that incorporates the Java developing kit *jdk1.5* and ant compiler. This tool is one with powerful graphic functions. But not only that, this tool can make links with external databases, web and embedded applications using secure web pages, but the main property is that it has an intelligent code editor which assists the programmer to each step and which points out in real time, without a pre-compilation of the code, the syntax code errors [5].

Java is a programming language which has native implementation of the parallel and concurrent calculus mechanisms. Real – time extension of this programming language is built, first of all, for a good and efficient work with the memory of the system, because the virtual machine Java is, in generally, a big consumer of memory [6].

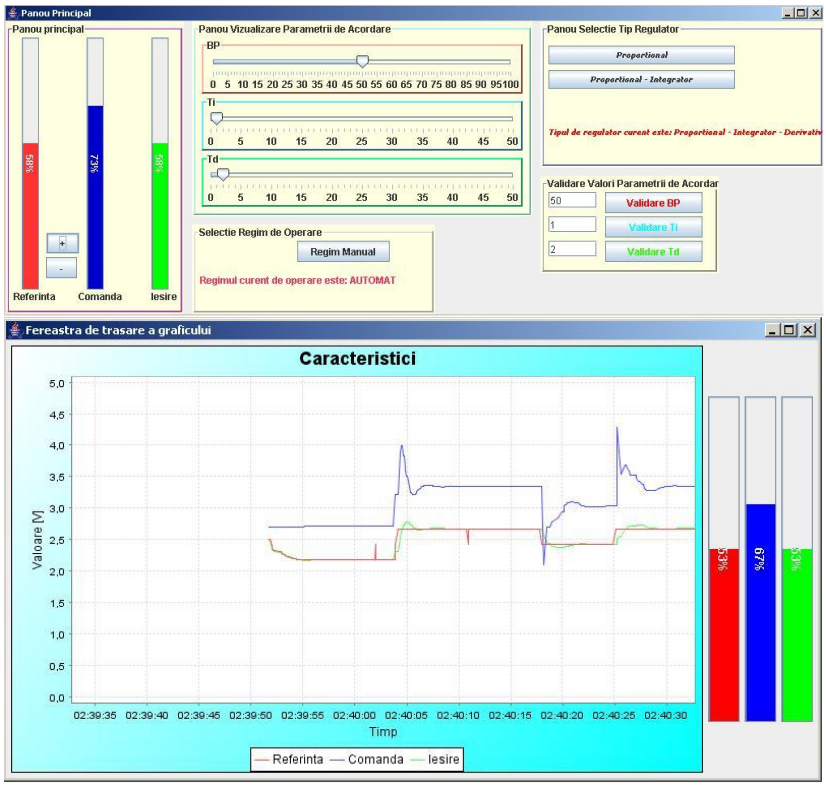

Fig. 2. Real – Time Flow Control Application Using Java Programming Language – while running.

The application is built with the object oriented programming technique, because the Java language is a pure object oriented programming language [7].

The base task has as objectives system initialization, variables initialization, launching the control task, implements methods for access to the private variables of the system (these variables can be used and modified only through these methods). This task is also responsible with the link between the different classes of this application. This task is implemented in the controlling class which resides in the file *controlling.java*.

The control task is the task that makes the command calculus based on the reference value and the value of the measured parameter from the process. This task has two critical sections: one in

the beginning, when the values of the controller parameters (the type of controller used: proportional, proportional – integrative and proportional – integrative – derivative, current operating regime) and the values of the reference, command and the measured parameter from the process) are updated, and one in the end when these values are updated with the values generated after the calculus has been made. This task is implemented in the control class which resides in the file *control.java*.

On-screen display task realizes the on-screen display of the three parameters of the system (reference, control and measured parameter); updating these values on the *JProgressBar* objects associated, update of the controller parameters, update of type of controller used and of the operating regime , when these parameters are updated by user. This task has a critical section, in it's beginning, when the current values are updated from the critical section, in order to make the on-screen display. This thread is implemented in display class, which resides in the *display.java*.

The mutual exclusion of the two tasks is made using a binary semaphore implemented in a class that is derivate from class general semaphore [1].

Real – Time Flow Control Application Using QNX Operating System

C/POSIX standard programming has as advantage explicit threads implementation by the developer. The big disadvantage of this solution is that the graphic user interface construction must be done using the tools of Photon kernel. This is done hard and complicated because it is required from the developer to define all the objects parameter used in command line code (writing code and call of functions). This is a big problem for application developer when a dead – line must be achieved. QNX system developers proposed later a solution for this, and starting with 3.x, and then with 4.x, they created an integrated development environment called Photon Application Builder (PhAB). This environment offers a wide graphic object palette, but the most important thing is that it is designed by the concepts event – driven and time – driven [4].

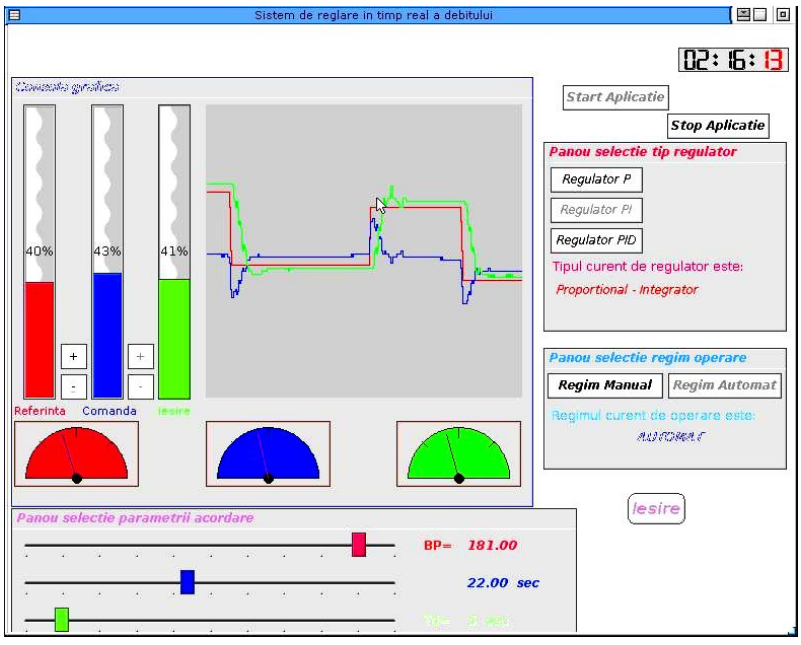

Fig. 3. Real – Time Flow Control Application Using QNX Operating System – while running.

Event – driven means that a code sequence is run as a result of an action (callback function). Therefore, to each object in the palette are assigned callback functions which contain code sequences that will be run when associated events take place. For example, a button object has as callback function an incremental value for a counter, which is associated to the event given by its press.

The big disadvantage of this integrated development environment is that it does not possess the so called characteristic thread – safe. This means that explicit thread implementation using

C/POSIX standard functions can and will make the system unstable or can block the computer[1].

Therefore, because the programming under C/POSIX standard is difficult when we develop the graphic user interface and because the PhAB puts at out disposal on object called Timer, which can be used for synchronization, we will discuss in the following part about the implementation under PhAB.

The application is designed to run on time – condition. In other words, the application will run some code sequences on cyclic periods of time. This is made using an object called PtTimer and it will ensure cyclic code execution on certain periods of time. The only restriction is that the cyclic code execution must be done between two consecutively timer interrupts. The moment in which timer interrupts occur is given by the PtTimer object attribute Pt_ARG_TIMER_REPEAT. This attribute can have integer values and it represents periods with the order of milliseconds. When we speak about computers with high speed processor and regardless of the process dynamics, it is ensured that the code sequence will run completely between two consecutively timer interrupts, even if this period is very short (for example 50 milliseconds).

Conclusions

On this paper we have shown both advantages and disadvantages of each implementation studied.

If we consider the example in this paper, we will see that for developing a simple application using CTask kernel, the programmer must write a lot of code and the graphic user interface is not spectacular. This is another great disadvantage because the beneficiary of the application wants a spectacular graphic user interface, even if the other solution is better but does not have a good graphic user interface.

Programming language Java and implicitly virtual machine Java are considered to be the tools "en vogue" from the most software application developers. They permit developing graphic user interfaces with highly appreciation from public, their integration in web sites and the link with external databases and the implementation of hierarchical control systems very complex. The greatest advantage of this solution is that by compilation it is achieved a higher application security. Another great advantage is represented by the Sun's motto: "Write once, run anywhere" [7]. Because of the portability of java machine on any operating system, the solution presented will run on any computer, regardless of the operating system, if the virtual machine Java is installed.

The big disadvantage of this solution, and implicitly the one of the Java virtual machine, is that for a good execution the machine need processors with very high speed performance.

One big advantage of QNX real – time operating system is that it permits to the programmer access to low level resources, leaving entirely to the programmer to develop applications in that manner that he considers to be reliable and secure.

Regarding the solution presented in this paper, QNX real – time operating system is the best of the three real – time programming platforms by the point of view of the system stability and of the level of resources consumption. On the period of testing the applications, the solution implemented under QNX real – time operating system have had a RAM memory consumption under 50%, the only moment in which that level was reached was at application start. Also, in that moment, the activity of microprocessor increased, but for a very short period of time. Overall RAM Memory consumption during the application run was under 20% and the microprocessor activity under 15%.

Because the QNX contains implementation of general drivers, it is not required for additional implementation or installing software for those drivers, not like Java virtual machine which requires driver implementation, and this helps saving disk space and a more efficient execution.

This operating system has native drivers implemented for working and communicating with embedded systems with microcontrollers.

Concluding, for complex control systems and for hierarchical and distributed control systems implementation, and because QNX has powerful networking use facilities, QNX real – time operating system is the best way to implement such solutions.

References

1. D r ă g o i c e a , M . – *Real–Time Systems and Programming Languages*. Ed. Printech, Bucharest, 2003.
2. P a r a s c h i v , N. – *Real–Time Application Engeneering. Lecture Notes*. Petroleum – Gas University of Ploieşti, 2005.
3. W a g n e r , T . – *CTask A Multitasking Kernel for C ver. 2.2*. 1990.
4. * * * – *QNX interactive help and www.qnx.com.*
5. www.netbeans.com.
6. www.rtj.org.
7. www.sun.com.

Privind disponibilităţile platformelor de programare în timp real CTask, QNX şi Java

Rezumat

Această lucrare îşi propune să realizeze o analiză critică a disponibilităţilor platformelor de programare în timp real CTask, Java şi QNX pe baza unui studiu de caz. Studiul de caz este reprezentat de proiectarea şi implementarea unei aplicaţii de timp real, care să realizeze procesul de reglare a unui debit. Algoritmul de reglare folosit în această lucrare este algoritmul clasic de reglare PI. În această lucrare nu s-a luat în discuţie echipamentele de automatizare (traductoare, elemente de execuţie, plăci de achiziţie), preocuparea principală fiind numai aceea de dezvoltare şi implementare a unei aplicaţii de conducere în timp real.