REAL-TIME PROGRAMMING

Hands-on « Windows XP temporal behaviour »

January 29, 2007

Objectives

This set of experiments will allow you to observe some aspects of the temporal behavior of Windows XP.

Tools and general advises

You will use two PCs. On the first one, you will execute your programs (watched machine). The second will be used as a logic analyser. Both are linked by a cable with 2 25pin plugs. On side of the cable is directly plugged in the parallel port of the watching PC. The other side is plugged into a small board which is itself plugged into the parallel port of the watched PC. The analyzer PC should have an odd identification number (xx on small sticker on the top right of the monitor with ICBC07PCxx). The other PC should have an even number.

The necessary files are located in the directory \\icwfile\Studdata\RealTime\. You just need to copy the ManipsNTTR directory and all its content to the root (C:) of the computer you use. Inside ManipsNTTR, there is one directory for each experiment in which all the necessary files are located. File ManipX.sln is the Visual .NET file of the experiment.

RTLauncher (in C:\ManipsNTTR\RTLauncher) is an application that will allow to start an application at real-time priorities. It is simple to used. Just lauch it by double clicking on its name. A window will open with a prompt to enter the name of the application to launch at real-time priorities. Type the name of your experiment (i.e. manip1 for the first one) and hit return. There is no need to specify the location of the application, as long as it is located in the RTLauncher directory, because by default RTLauncher will look in the RTLauncher directory. The Visual .NET projects have been set so that the application generated by builds will be located in that directory.
To avoid possible blockings of your PC, please always check your application by executing it directly (not from Visual Sudio) before launching it with RTLauncher. Once launcher with it, an application cannot be halted.

To start the logic analyser, you need to reboot the PC and press F8 at the count down. A menu will appear with a number of boot possibilities. Choose « boot disquette temps reel ». The system will then start in DOS mode. As soon as you get the prompt, type "logic.exe". This will start the analyser.

With the analyser, you can visualise the state of 5 of the lines of a PC parallel port. In the programs that you will use or create, you will use invocation of WriteParallelPort. The second parameter of this procedure contains the value that is to be written at the output. The table below gives you the match between the value, the parallel port lines and the analyser signals.

Value in C++	Data line on parallel port	Channel of the analyser
0x01	0	1
0x02	1	0
0x04	2	2
0x08	3	3
0x10	4	4

With the analyser, you may choose the following parameters:

- The sampling frequency (sample rate) in numbers of sample per second (R or r command). Most of the time you will use the maximum value for the experiments.
- The triggering condition (T or t command)
- The (trigger input) channel on which the triggering condition is applied (I or i command)
- The number of samples (duration) during which the signal should remain unchanged after the triggering condition has been found (d or D command)

Each measurement is started using the S or s (start/stop) command. The analyzer will then tell you if the triggering condition has been found. As soon as the number of samples is sufficient to fill the memory buffer, the "buffer full" message is displayed. You may stop the acquisition before by hitting the S (or s) key. The content of the buffer can then be displayed using the V or v (view trace) command. The menu then gives you a number

of options to change the time scale and the starting point of the display. Please note that as soon as the buffer is full, you may not start any new measurement unless you clear the buffer (c or C command).
It is possible that the analyzer program freezes (sorry). Your only option is to reboot the PC and start again as indicated above.

Experiment 1 :

Use project «Manip1 » in directory « C:\manipsNTTR\manip1 ». This project implements an application that starts a thread running a loop that increments a counter. The objective is to observe the difference in behavior is the application is executing at real-time priorities or at non real-time one.

- Look at the structure of the program and the primitives used to create and start a thread.
- Execute the program directly (without RTLauncher). What happens ?
- Execute the program with RTLauncher. What happens ? Why ?

If the message « Could not open Parallel Port! » is displayed, this means that the parallel port driver has not been properly (or not at all) installed. In such a case, copy the file portio.sys (whihc is located in the directory ManipsNTTR) to the directory C:\WINDOWS\system32\drivers\ then localize the file TempReelPortIo (in the same directory as portio.sys) click on that file using the right mouse button and use the « merge » command. You should then reboot the computer and everything should be fine.

Experiment 2 :

Use project «Manip2 » in directory « C:\manipsNTTR\manip2 ». This program starts a thread that executes repeatedly a loop in which line 2 of the parallel port is toggled. The objective is to observe that your thread sometimes stops executing for durations up to a few milliseconds. You will now need the logic analyzer (analyzer parameters: rate max, input 2, level 1, duration 20).

- Watch the structure of the program and the primitives that are used.
- Look at the thread execution function and predict the behavior.
- Execute the program directly (without RTLauncher). What happens ? What do you observe on the analyzer ?
- Setup the analyzer so that it triggers only if line 2 of the parallel port does change its state during more than 1ms and execute again the program directly. What do you observe ?
- Execute the program with RTLauncher. What happens ? Why ?

Experiment 3 :

Use project «Manip3 » in directory « C:\manipsNTTR\manip3 ». The application starts 2 threads that share the access to the parallel port in an exclusive manner using a semaphore. The objective is here to observe the policy used by Windows XP to manage the queues of tasks trying to gain access to a semaphore. The analyzer will tell you which is the last executed thread by looking at the state of the parallel port lines 2 and 3 at the end of the program execution. If line 2 is at one, thread1 was the last thread to execute. If line 3 is at one, thread 2 was the last one. You will have to find out the semaphore grant policy (analyser parameters: rate 50000, input 3, rising edge, duration 1)

- Watch the structure of the program and the primitives that are used.
- Look at the thread execution function and predict the behavior.
- Execute the program directly (without RTLauncher). What happens ? What do you observe on the analyzer ?
- Execute the program with RTLauncher. What happens ? Why ?
- Try to change the priorities of the threads and check if there is any change ?
- Try to change the order in which the threads request the semaphore and check again ?

Experiment 4 :

Use project «Manip4 » in directory « C:\manipsNTTR\manip4 ». This project contains an application that starts 3 threads. Two of these (thread1 and thread2) share a semaphore to manage the access to the parallel port. The third thread (thread3) simulates the behavior of a task with intensive calculations (a large loop that does not end). All three execute at the same priority. The objective of this experiment is to observe the behavior of the time slicing (quantum) in Windows XP. (analyser parameters: rate 10000, input 2, rising, duration 1)

- Watch the structure of the program and the primitives that are used.

- Look at the thread execution function and predict the behavior.
- Execute the program directly (without RTLauncher). What happens ? What do you observe on the analyzer ?
- Execute the program with RTLauncher. What happens ? Why ?
- Modify the program to achieve a predictable behaviour.

Experiment Multithreading :

You will now observe the influence of the multithreaded Pentium architecture on the temporal behavior. First you need to swap the roles of your 2 PCs. Please reboot, the odd number PC with Windows XP. At the same time, restart the even number PC with the analyzer (see instructions above). You will have to swap the parallel cable by putting the small PCB on the side of the PC running Windows XP. This one now runs with the multithreading option enabled:

- Reuse experiment 1 and execute again manip1.exe using RTLauncher. What is the difference with what you observed previously ? Please explain.
- Add in the program a second thread that you run at priority 31. What happens if you execute again the program under RTLauncher ? Why ?
- Reuse experiment 3 and execute again manip3.exe. What is the difference with what you observed previously in experiment 3 ? Please explain.
- Reuse experiment 4 and execute again manip4.exe with and without RTLaucher. What is the difference with what you observed previously in experiment 4 (without multithreading) ? Please explain.

Experiment 5 :

The set of experiments from 5 to 8 will lead you a manner to implement a real-time application with predictable behavior under Windows XP. You may freely use the parallel port and the analyzer to demonstrate the behavior of your application. We advise you always to compare the behavior under real-time and non real-time priorities. You may be surprised. Use project «Manip5 » in directory « C:\manipsNTTR\manip5 ». Use this project to create an application that has a dispatcher thread that is started periodically and controls the execution of other periodic threads.

Experiment 6 :

Modify the project of experiment 5 so that the dispatcher activates the other periodic threads according to their relative priorities.

Experiment 7 :

Modify the project of experiment 6 so that the dispatcher leaves some time for the rest of the OS threads to execute. This should demonstrate when you execute with RTLauncher. The display should no longer be frozen.

Experiment 8 :

Suggest a way to realize a dispatcher that implements the EDF policy.