Task scheduling and control

LUMINITA GIURGIU, MIRCEA POPA
Technical Sciences Department
"Nicolae Balcescu" Land Forces Academy
Revolutiei no. 3-5 Street, Sibiu
ROMANIA
http://www.armyacademy.ro

Abstract: - This paper demonstrates some aspects of task scheduling policy effects on embedded control systems. This problem is studied because almost all control algorithms are realized by computers and controllers are often implemented as one or several tasks on a microprocessor with a real time operating system. The case of three tasks running concurrently on the same CPU and controlling three different dynamic systems is simulated in MatLab TrueTime toolbox environment. Embedded control systems are subject to limited computer resources that are in fact shared resources, for which the tasks compete. A priority based approach and a dead line based scheduling are confronted in order to establish advantages and disadvantages.

Key-Words: - scheduling policy, computer based control, embedded control systems

1 Introduction

Embedded control systems study requires inter-disciplinary knowledge of both control engineering and computer science and their inter-relations.

1.1 Scheduling policy

Real time scheduling theory deals with the problem of, given a set of tasks, finding an execution order that assures that all tasks meet their timing constraints. Scheduling algorithms fall in two categories: static and dynamic scheduling.

Static scheduling is an offline approach: an optimized execution order is predetermined and this execution order is cyclically repeated at runtime.

Dynamic scheduling is an online approach: the running task decision is taken at runtime.

In scheduling theory a task model is considered, where all tasks are periodic and where each task, i, is characterized by the following parameters: a fixed period, T_i, a hard deadline, D_i, and a fixed and known worst case execution time (WCET), C_i.

The mechanism supported by all major commercial real time operating systems is the preemptive fixed priority scheduling where each task is assigned a fixed priority value. During runtime, the ready task with the highest priority gets access to the CPU, meaning that if a task with a lower priority is currently running, this one is preempted by the higher priority task.

It is somehow natural to assume that for control tasks their relative deadlines, D_i, are equal to their periods, T_i. The most common priority assignment is the rate-monotonic (RM) assignment, where the priorities are set according to the periods of the tasks: the shorter the period, the higher the priority.

An alternative approach based on the absolute deadlines of the tasks is earliest-deadline-first (EDF) scheduling which exploits dynamic priority assignment: the task with the shortest remaining time to its deadline will get access to the CPU at any point in time.

1.2 Control implementation

In a computer-based control system the continuous process output is sampled at regular time intervals and converted to digital form by an A/D converter. The control algorithm reads the sampled process output and computes a control signal that is converted by a D/A converter back to analog form.

A periodic control loop implementation is shown in the Table1 pseudo code.

Table 1

```
t = currentTime();
LOOP
        Read Inputs;
        Control Compute (first part);
        Write Outputs;
        Update Internal States (second part);
        t = t + period;
        waitUntil(t)
 END
```

The reading of inputs and writing of output signals correspond to direct calls to external A/D

and D/A conversion interfaces and it is also possible to have the sampling and actuation being performed by dedicated tasks, when buffers are used to communicate the values between the tasks.

It is important to observe that a controller task implementing a control loop (the control algorithm) is divided into two parts in order to minimize the input output delay: the first part computes the control signal based on current measurements and previous states, and the second part then updates the internal states of the controller for the next sample. Input-output latency is caused by preemption from higher priority tasks and by the execution time of the control algorithm itself.

2 The model of the control system

The case under study takes three control tasks running concurrently on the same CPU, controlling three different dynamic systems. The three systems under control are second order system with different dynamics: slow, middle and fast dynamics. The corresponding controller task's periods are different according to the system's dynamics: the faster the dynamics, the shorter the period of the task.

The TrueTime computer block is connected with ordinary Simulink blocks to form a real time control system, see figure Fig. 1.

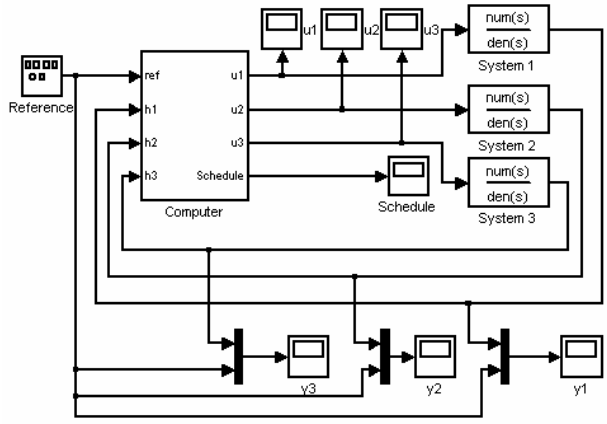

Fig. 1 The control system Simulink model

The control system model will be used in simulations in order to demonstrate the effect of the applied scheduling policy on the global control performance. It is also studied the input-output latency effect on the overall system stability.

3 Tools and simulations

There are numerous tools that support simulation of control systems or simulation of real time scheduling, but very few tools support co-simulation of control systems and real time scheduling.

The True Time simulator is a complete co simulation tool based on MATLAB/ Simulink and in its current version it supports task scheduling by arbitrary scheduling policies, network simulation by standard MAC layer protocols, and a variety of real time primitives used for experimentation with flexible scheduling and compensation schemes.

The three systems are controlled by controller tasks implemented in a TrueTime kernel block. Before a simulation can be run, it is necessary to initialize kernel block and to create tasks for the simulation. The execution of tasks is defined by code functions.

A code function is further divided into code segments according to the execution model. All execution of user code is done in the beginning of each code segment. The execution time of each segment should be returned by the code function.

The kernel is initialized by the initialization script, providing the number of inputs and outputs, the scheduling policy and creating periodic tasks that uses defined code functions.

In the initialization script the scheduling is specified by the function ttinitkernel and the task are created with ttCreatePeriodicTask function who specifies the name of the associated code function.

A sample of code function is given in Table 2.

Table 2

```
function [exec_time, data] = code_f1(seg, data)
switch seg,
 case 1,
   % read reference
   r = ttAnalogIn(data.rChan);
   % read process output
   y = ttAnalogIn(data.yChan);
   % calculate control signal
   [data.ctr, data.u] = calc_output(data.ctr, r, y);
   exectime = 0.006;
 case 2,
   ttAnalogOut(data.uChan, data.u);
   data.ctr = update_states(data.ctr);
   exectime = 0.002;
 case 3,
  exectime = -1; % task execution finished
end
```

Running the simulation of the control system, seven graphics are obtained: three graphics for the control signals u1, u2, u3, three graphics for the reference and outputs signals y1, y2, y3 and the schedule graphic of the tasks during the simulation. Figure Fig. 2 shows the schedule graphic of the

tasks during the simulation and figure Fig. 3 shows the output signal y3 compared with the reference signal, in the EDF case.

Fig. 2 Schedule graphic of the tasks: task1 (blue), task2 (green), task3 (red)

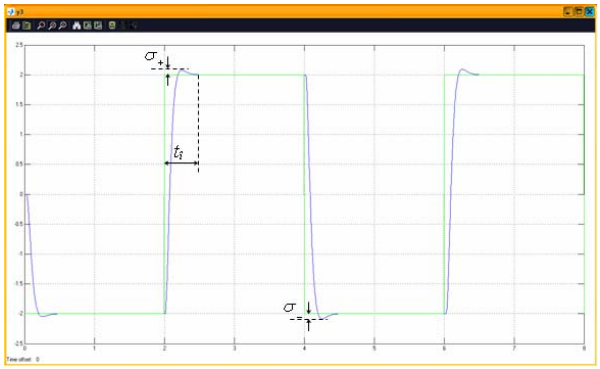

Fig. 3 Output signal y3 (blue) compared with reference signal (green)

- although the RM scheduling is a fixed priority policy, the slowest task may be active twice a period;
- in the EDF case, even if the tasks miss their deadlines, the overall control performance remains satisfactory.

In order to study the input-output latency effect, the execution time of the first segment in the associated function code of the task was gradually increased and the following results were obtained:

- in the RM case, beginning with the 0.010 s value of the execution time, only the first system becomes unstable and the second system presents 15-20% values of the overshoots, see figure Fig. 4;
- in the EDF case, beginning with a larger value of the considered first segment execution time, 0.016 s, all the systems become unstable.

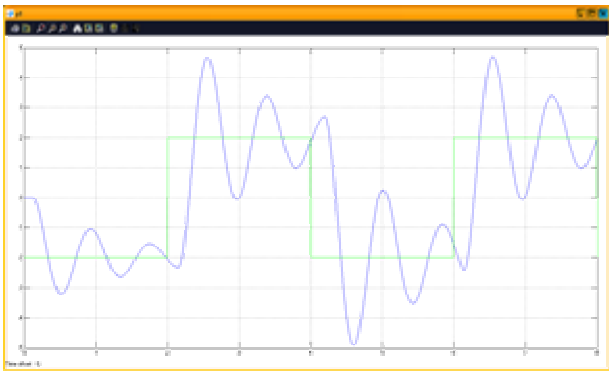

Fig. 4 Unstable system1 (y1)

3.1 Results

The simulations have considered the RM and the EDF scheduling policy cases. The observed characteristics in the graphics of the output signals where: positive front overshoot σ_+ , negative front overshoot σ_- and the reference following delay t_i.

Studying the computer schedule and the control performance for these cases, the following are noticed:

- the three control loops are stable;
- the tasks meet their deadlines;
- in the RM case the best characteristics are obtained for the third system controlled by the task with the shortest period:

  σ_+ =4,63%, σ_- =4,6%, t_i =0,43 s;
- in the EDF case the observed characteristics are approximate the same for all the three output signal y1, y2, y3, demonstrating that the dynamic assignment of the task priorities assures an equilibrate performance of the entire embedded control system:

  σ_+ =4,1%, σ_- =4,1%, t_i =0,4 s;

4 Conclusions

Undertaken studies and simulations and the results obtained have led to some conclusions and considerations regarding not only the advantages and disadvantages of applied scheduling policy, but also regarding the technique of the simulation. The work carried out is also highlighted by some considerations about author's contributions.

In order to assign priorities, global information about the relative importance of all tasks in the system is needed, which is not required to assign deadlines. So, a first benefit of dead line based scheduling over priority based scheduling is that it is more intuitive to assign deadlines to tasks than to assign priorities.

The main drawback with EDF is that it offers no guarantees at all during overload. In that case all tasks will miss their deadline, (the domino effect), and for hard real time systems this may be fatal. However, the result during overload under EDF is that the effective periods of the tasks will be scaled in such a way that the utilization of the system is

still 100 per cent, so under reasonable overload, for most control systems this fair distribution of resources still give reasonable performance for all control loops.

A priority based approach favors high priority tasks over low priority tasks, with the possible consequence that low priority tasks may not execute at all, being starved. Using dead line based scheduling instead, the available CPU time is distributed among the tasks in a more fair way and better results concerning the performance and stability of the control loops where obtained. Depending on the application this may or may not be a desired feature.

The major drawback with dead line based scheduling is the lack of commercial products supporting it.

The simulation of the execution code representing the control algorithm, presents three special aspects: the simulation was run at segment code level, not at instruction level, each segment may had a different execution time and the task interacts with other tasks or with the environment, only at the beginning of the each code segment.

The control system Simulink model and the simulation presented method are also suitable to other scheduling policies like FP (fixed-priority) and DM (deadline-monotonic), by adapting the initialization script with appropriate init function parameters in the kernel block.

The systems under control are chosen with different dynamics: slow, middle and fast dynamics in order to study the appropriate corresponding controller task's periods, which are different according to the system's dynamics: the faster the dynamics, the shorter the period. By changing the sampling period, resulting control performance is revealed.

The simulations were run verifying that the controllers behave as expected and the effect of different input-output delays were simulated by gradually increasing the execution time of the first segment of the associated code function.

Because of the fact that in embedded control systems the tasks compete on shared computer resources, two scheduling policies where applied and the corresponding advantages and disadvantages where established based both on observed characteristics in the graphics of the output signals compared with the reference and the schedule graphic of the tasks. The simulation presented method demonstrates that the dynamic assignment of the tasks priorities assures an equilibrate performance of the entire embedded control system and supports larger values of the considered first segment execution time in the associated function code of the task.

The scheduling and execution of control tasks is simulated in parallel with the continuous process dynamics, demonstrating that the initialization script is the simulation's required piece, user defined and supported by the TrueTime kernel block, that really joins the two aspects and problems involved: task scheduling and control.

References:

[1] Henriksson, D., A. Cervin, and K.-E. Årzén, *True-Time: Simulation of control loops under shared computer resources*, Proceedings of the 15th IFAC World Congress on Automatic Control, Barcelona, Spain, 2002.

[2] Åström, B. Wittenmark, *Computer Controlled Systems*, Prentice Hall, Upper Saddle River, N.J., 1997.

[3] Cervin, A., *Integrated Control and Real-Time Scheduling*, PhD thesis ISRN LUTFD2/TFRT--1065--SE, Department of Automatic Control, Lund Institute of Technology, Sweden, 2003.

[4] Årzén, K.-E., *Real-Time Control Systems*, Department of Automatic Control, Lund Institute of Technology, Lund, Sweden, 2001.

[5] Liu, J. and E. Lee, *Timed multitasking for real-time embedded software*, IEEE Control Systems Magazine, 23:1**,** 2003.

[6] Ohlin, M., Henriksson, D., Cervin A., *TrueTime 1.5—Reference Manual*, Manual, Department of Automatic Control, Lund University, Sweden, January 2007.