A Simple Real-Time Scheduler

Mattsson, Sven Erik

1978

Document Version:
Publisher's PDF, also known as Version of record

Link to publication

Citation for published version (APA):
Mattsson, S. E. (1978). *A Simple Real-Time Scheduler*. (Technical Reports TFRT-7156). Department of Automatic Control, Lund Institute of Technology (LTH).

Total number of authors:
1

CODEN: LUTFD2/(TFRT-7156)/1-010/(1978)

A SIMPLE REAL-TIME SCHEDULER

SVEN ERIK MATTSSON

Department of Automatic Control
Lund Institute of Technology
November 1978

Dokumentutgivare
Lund Institute of Technology
Handläggare Dept of Automatic Control
Sven Erik Mattsson

Författare
Sven Erik Mattsson

Dokumentnamn
REPORT

Utgivningsdatum
Nov 1978

Dokumentbeteckning
LUTFD2/(TFRT-7156)/1-010/(1978)

Ärendebeteckning
06T6

10T4

Dokumenttitel och undertitel
18T0
A SIMPLE REAL-TIME SCHEDULER.

Referat (sammandrag)

This paper describes a simple real-time scheduler for process control applications. The scheduler was designed so that a small computer without a real-time operating system could be used to control a small process.

A top-down outline has been attempted to show the solution. The scheduler is implemented on LSI-11 under the assumption that the programs are written in Pascal and compiled by the OMSI Pascal compiler, but it will probably be easy to modify the implementation for other computer systems.

The scheduler has been used with success in a laboratory course.

Referat skrivet av
Author

Förslag till ytterligare nyckelord
44T0

Klassifikationssystem och -klass(er)
50T0

Indextermer (ange källa)
52T0

Omfång
10 pages

Övriga bibliografiska uppgifter
56T2

Språk
English

Sekretessuppgifter
60T0

ISSN
60T4

ISBN
60T6

Dokumentet kan erhållas från
Department of Automatic Control
Lund Institute of Technology
PO Box 725, S-220 07 Lund 7, SWEDEN

Mottagarens uppgifter
62T4

Pris
66T0

1. INTRODUCTION

This paper describes a simple real-time scheduler for process control applications. The scheduler was designed so that a small computer without a real-time operating system could be used to control a small process or a part of a process.

Discussions between Hilding Elmqvist, Leif Andersson and myself eventually led to a simple but still powerful solution. A top-down outline has been attempted to show the solution. The scheduler has been used with success in a laboratory course and this application is described in Andersson, Åström (1978).

2. PURPOSE AND DESIGN

The application that inspired this real-time scheduler was a small process control system for educational use. The computer in this system is an LSI-11. The hardware is described in Andersson (1978).

The computer is expected to control a process and to communicate with a human operator. This indicates a structure with two interacting concurrent programs. The first program, here called FG (ForeGround), should control the process and has to be run periodically. The second program, here called BG (BackGround), should communicate with the operator and act at his request.

The execution time of FG can be regarded as short and it is more important to control the process than to communicate with the operator, so it is reasonable to design a foreground-background, run-to-completion scheduler. This means that, whenever FG wants to run, it is allowed to do so and BG has to wait until FG has completed its execution.

In the implemented version of the scheduler it is assumed that the computer is an LSI-11 and that FG and BG are written in Pascal, but this suggested scheme will work with other types of hardware and software as well.

Following the advice

> "A creative programmer will try to use a particular application as an inspiration to look for program structures that can be used in a class of similar applications."

in Brinch Hansen (1977) the design problem is stated in the following way:

Problemstatement: Design a foreground-background, run-to-completion scheduler. It should

i) run two programs (FG and BG)
ii) run FG periodically with variable period time
iii) manage non-existing FG (it is assumed that BG never exits)
iv) preempt BG if FG is to be run and let BG wait until FG completes its execution.

If it is assumed that the variables have proper initial values, the first sketch of the clock-interrupt-routine in a pseudo-Pascal notation might look as follows:

```
procedure Clockinterrupt;
begin
  if period <=0 then icnt:=1
  else begin
    icnt:=icnt-1;
    if icnt=0 then begin
      icnt:=period;
      Savestatus;
      Run(FG);
      Restorestatus
    end
  end
end;
```

Algorithm 1 First Attempt

Period is a global variable, which contains the desired number of clock interrupt intervals between two runs of FG. The routines Savestatus and Restorestatus will be discussed later. The routine Run(FG) starts up and executes FG.

Up to now the real time properties of Clockinterrupt have not been discussed. Clockinterrupt is not an ordinary sequential procedure. If period is greater than one it is natural to allow the execution of FG to take more than one clock interrupt interval, but icnt must still be updated every clock interrupt. Hence, the clock interrupt must be enabled during Run(FG). Now the procedure Clockinterrupt divides in a natural way into three parts. Part I is up to Run(FG), part II is Run(FG) and part III is the rest of the procedure.

The clock interrupt signal can of course arrive during the execution of part III. As seen in Algorithm 1, part I and III have common variables and conflict situations might arise. A simple and reliable way to solve these possible conflicts is to make part I and III indivisible. In the final version it is done by disabling the interrupt. The time spent with the interrupt off is very short, typically 10 - 20 machine instructions.

What should happen if icnt=0 in the first part and FG is active? In many real-time systems a request on an active task is neglected, but in this application it is desirable to remember requests. We have, however, decided that only one request but no more is remembered. The request is stored in the variable lag.

```
procedure Clockinterrupt;
begin
  Disableinterrupt;
  if perod <=0 then begin
    icnt:=1;
    lag:=0 end
  else begin
    icnt:=icnt-1;
    if icnt=0 then begin
      icnt:=period;
      lag:=1;
      if not active then begin
        active:=true;
        Savestatus;
        repeat
          lag:=0;
          Enableinterrupt;
          Run(FG);
          Disableinterrupt
        until lag=0;
        active:=false;
        Restorestatus
      end
    end
  end;
  Enableinterrupt
end;
```

Algorithm 2 The Final Version

The procedures Savestatus and Restorestatus are machine dependent and in order to make the discussion simpler and clearer they will be discussed further in the implementation part. It will only be remarked here that it may be difficult or impossible to implement Savestatus and Restorestatus if FG and BG have common non-reentrant procedures. Further an interface routine is needed to start up Clockinterrupt and this is also discussed in the implementation part.

3. IMPLEMENTATION

This chapter shows an implementation when the computer is an LSI-11 and when FG and BG are written in Pascal and compiled by the OMSI Pascal compiler. If routines common to both FG and BG are reentrant and if it is possible to link assembly code to FG and BG, it will probably be easy to implement the scheduler on other computer systems.

The OMSI Pascal compiler allows external, global procedures so it is possible to write the scheduler in assembly language.

As seen from Algorithm 2 and the specifications, Clockinterrupt needs the address to FG and the value of period and it should be possible to change the value of period. If FG is defined as a procedure it is easy to implement Run(FG) and the procedure-name FG can be passed as a parameter to Clockinterrupt. If period is a global variable and if the address of period is passed to Clockinterrupt it is easy to change period from FG or BG.

Hence, declare period as a global variable of type integer and FG as a global procedure. Declare a global, external procedure Schedule in the following way:

```
procedure Schedule (procedure FG; var period: integer);
                    external;
```

Schedule starts up a periodic execution of FG and the period time depends on the value of period. The Line Time Clock (LTC), when enabled, interrupts every 20 ms. This means that the desired time between to executions of FG is 20* period ms. Schedule is to be found in Appendix 1. The OMSI Pascal compiler allows insertion of assembly code inline. It is possible to reference the Pascal variables from the assembly code.

The implementation of Clockinterrupt is straightforward. The Run(FG) call is very similar to an ordinary call of FG. The main difference is that the Run(FG) is decided in time, but an ordinary call of a procedure is decided by the place in the code. But this difference does not matter. The important issue is that FG is executed (as an ordinary procedure) until it exits. BG has information only in the registers, on the stack and in its variables. Because Pascal allows recursive procedures, the compiler produces reentrant code and all intermediate information is placed on the stack. Because FG acts as an ordinary procedure, Savestatus has only to save the registers and no more. SAVREG and RESREG are Pascal library routines, which save and restore the registers.

An OMSI Pascal program stores the address to the global variable area in $RESR5 and R5 (register 5), and this address is used by FG too. But when a clockinterrupt is received, it is not sure that R5 contains this address e.g. an external assembler procedure can temporarily use R5 for an another purpose, so the address of the global variable area must be stored in R5 before calling FG.

Schedule initiates all local variables of Clockinterrupt except for the variable active. Schedule can be called more than once e.g. with different FG parameters and an old FG may be active and it has to exit before a new FG is started up. The very first value of active is set to false (0) as seen from the declaration of active in Appendix 1.

Apart from the discussions above the code in Appendix 1 is selfexplaining and this completes the implementation.

As mentioned in the introduction an application can be found in Andersson, Åström (1978).

4. REFERENCES

Andersson, L. (1978): A Process Interface for the LSI-11
Dept of Automatic Control, Lund Institute of Technology, Lund, Sweden.
To appear.

Andersson, L., Åström, K.J. (1978): An Interactive MISO Regulator. Dept of Automatic Control, Lund Institute of Technology, Lund, Sweden.
CODEN: LUTFD2/(TFRT-7154)/1-034/(1978)

Brinch Hansen, P. (1973): Operating System Principles, Prentice-Hall Inc, Englewood Cliffs, NY.

Jensen, Kl, Wirth, N.(1975): Pascal-user and manual report, Second Edition, Springer Verlag, Berlin.

OMSI PASCAL-1 Documentation Version 1.1,
Oregon Minicomputer Software, Inc,
2340 SW Canyon Road, Portland, Oregon 97201

RT-11 System Reference Manual
Order No. DEC-11-ORUGA-C-D,DN1, DN2
Digital Equipment Corporation Massachusetts 1976.

APPENDIX 1

```
procedure Schedule(procedure FG; var period:integer);
begin
  {$C
        MFPS     %0                 ;Save current interrupt status
        MTPS     #^O340             ;Disable interrupt
        MOV      FG(%6),ADDRFG      ;Save address to FG in ADDRFG
        MOV      PERIOD(%6),APER    ;Save address to period in APER
        MOV      #1,ICNT            ;icnt:=1
        CLR      LAG                ;lag:=0
        MOV      #CLKINT,@#^O100    ;Set up the Clock interrupt vector
        MOV      #^O340,@#^O102
        MTPS     %0                 ;Restore interrupt status
  }
end;
{$C
;
;        INTERRUPT SERVICE
;
CLKINT:                             ;procedure Clockinterrupt;
                                    ;begin
                                    ;  Disableinterrupt;
        TST      @APER              ;  if period<=0 then begin
        BGT      RUN
        MOV      #1,ICNT            ;    icnt:=1;
        CLR      LAG                ;    lag:=0 end
        BR       RETURN
RUN:                                ;  else begin
        DEC      ICNT               ;    icnt:=icnt-1;
        BGT      RETURN             ;    if icnt=0 then begin
        MOV      @APER,ICNT         ;      icnt:=period;
        INC      LAG                ;      lag:=1;
        TST      ACTIVE             ;      if not active then begin
        BNE      RETURN
        INC      ACTIVE             ;        active:=true;
        .GLOBL  SAVREG,$RESR5
        JSR      %7,SAVREG          ;        Savestatus;
        MOV      $RESR5,%5
1$:                                 ;        repeat
        CLR      LAG                ;          lag:=0;
        MTPS     #0                 ;          Enableinterrupt;
        JSR      %7,@ADDRFG         ;          Run(FG);
        MTPS     #^O340             ;          Disableinterrrupt
        TST      LAG                ;        until lag=0;
        BNE      1$
        CLR      ACTIVE             ;        active:=false;
        .GLOBL  RESREG
        JSR      %7,RESREG          ;        Restorestatus
                                    ;      end
                                    ;    end
                                    ;  end;
RETURN:                             ;  Enableinterrupt
        RTI                         ;end;
;
ACTIVE: 0
ADDRFG: 0
APER:   0
ICNT:   1
LAG:    0
```