Event driven multitasking - a syntax

Jens Zander

SECTRA-Colon Systems AB
P.O. Box 9104
S-580 09 Linköping
SWEDEN

ABSTRACT

In many real-time programming applications, programs are to be initiated at the occurence of some internal or external event. Examples of such events are the real-time clock reaching some predetermined value, an I/O device needing service or some logical condition being fulfilled. The classical way of solving some of these problems is to use hardware interrupt signals. In this paper, situations are investigated where for various reasons interrupts cannot be used. An example is when the condition tested is a very complex one. A FORTH-syntax for general event handling is proposed, including the structures EVERY, AFTER and WHENEVER .... PERFORM. An implementation for (time-shared) multitasking FORTH systems is sketched.

1. Introduction

The microcomputer has become an invaluable tool in automatic control and real-time data-aquisition systems. Many industrial control applications of fair complexity are more or less 'one-of-a-kind' systems demanding a considerable amount of dedicated program development effort. For these applications FORTH systems with added real-time tools constitute an excellent programming and operating environment. Not only program development but also the installation, tuning and continued operation benefits from an interactive environment.

Real-time control systems can basically be seen as event-driven systems. By this we mean that the occurance of certain external events trigger certain responses from the control system. This is very much different from ordinary programming where the order of all actions is determined by the program flow (1). As a few examples of external events we have:

a) The operator pushing the 'TEST'-button

b) The temperature in some vessel has become to high

c) The real-time clock passes midnigth

To these events the control system may respond by:

a) Initiating the system test-function

b) Displaying an alarm-message

c) Increasing the heater power (if the electric power is cheaper after midnight)

In typical control applications, the control system has to be capable to handle up to hundreds or thousands of such events. These events may not occur very often, but nevertheless, for each of them a specific action has to be defined. Further all these events occur at unpredictable instants of time.

The classical approach to real-time programming is to use hardware interrupts to signal the occurence of an event. Each event has its interrupt service routine. The advantage of this method is the fast response; an event is handled within microseconds. The main disadvantage is that all events that are to be recognized by the system have to be defined when designing the hardware of the system. Complex event definitions naturally give rise to complex (and inflexible) hardware. Instead, event conditions can be checked in software, thereby sacrifying some of the short response time. Doing this using conventional FORTH programming techniques gives a poor result with complex definitions and hardly any possibilities to alter the program flow interactively. In the following we describe one possible event handling syntax that cures some of the drawbacks of the software approach. Finally we sketch an implementation.

2. An event handling word set

AFTER (d --- n) I

Used in the form:

```
d  AFTER name
```

Searches the context vocabulary after the word 'name' and schedules it for execution after d time units. Returns the unique ID n of this event.

```
EVERY          ( d --- n )                             I
```

Used in the form :

```
     d EVERY name
```

Searches the context vocabulary after the word 'name' and schedules it for repeated execution every d time units. Returns the unique ID n of this event.

```
WHENEVER       ( --- )        Compiling                I,C
               ( d --- n )    Executing
```

Used in the form:

```
     : xxx ... WHENEVER .. ( condition )
               PERFORM  .. ( action )  ;
```

When xxx is executed the (condition) part is executed every d time units. If the condition part results in a true flag, execute (action) part of the definition. xxx will also return the unique ID n of this event.

```
PERFORM        ( --- )        Compiling             I,C
               ( f --- )      Executing
```

Used in WHENEVER...PERFORM construct to mark start of the action part. Cf WHENEVER.

```
EHALT          ( --- )                              C
```

Stop the repeated execution of an EVERY- or an WHENEVER PERFORM event. Use in event definition only.

```
ESTOP          ( n --- )
```

Remove event with ID n.

```
ESTAT          ( --- )
```

Display all pending events.

```
SECONDS        ( n --- d )
```

Convert the time interval n into time units.

As can be seen from the word set, the real-time clock is the basic event generating element in the event handling syntax. Nevertheless, we handle both time-oriented (EVERY and AFTER) and condition-oriented events (WHENEVER...PERFORM). The latter type is implemented by using time-oriented events. This increases the workload of the system (we have to check a lot of event conditions) but gives us a way to define priorities among the handlers of different events. The time-interval d determines the rate by which the conditions are to be checked and thereby the response time of the event service routine. To clarify these concepts let have a look at a few simple examples:

a)
```
15 SECONDS AFTER SWITCH-ON
25 SECONDS AFTER SWITCH-OFF
```

We schedule the word 'SWITCH-ON' to be executed after 15 seconds and the word ' SWITCH-OFF' to be executed after 25 seconds.

b)
```
3 SECONDS EVERY TOGGLE-LED
```

Executes the word 'TOGGLE-LED' every 3 seconds. May be used to flash a LED on the control panel.

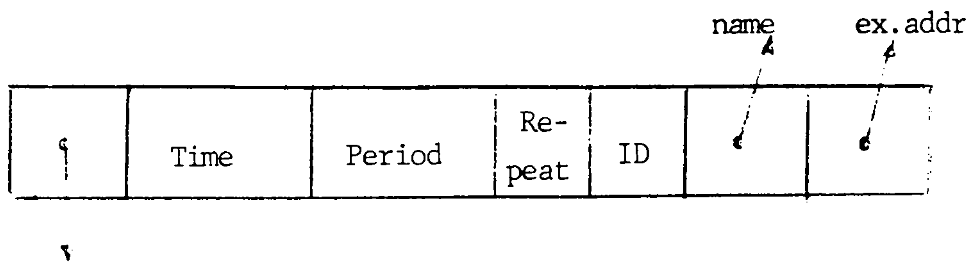
c)
```
: CHECK-TEMP  ( --- n )
     10 SECONDS
     WHENEVER
          TEMPERATURE TEMP-LIMIT >
     PERFORM
          TEMP-ALARM
;

CHECK-TEMP DROP
```

We define the word 'CHECK-TEMP' that will start to continously monitor the value returned by the function 'TEMPERATURE'. As long as this value is larger than the constant 'TEMP-LIMIT' the word 'TEMP-ALARM' is executed. The final 'DROP' discards the event ID not used here.

By using the words AFTER, EVERY and the WHENEVER..PERFORM construct, we may now interactively define (almost) any number of events and the corresponding actions. The events are handled as concurrent processes and may be monitored with the ESTAT command, showing all current events. Events may also be removed using the ESTOP command.

3. Implementation

In the following we outline one possible implementation. The implementation is centered around the event queue. The event queue is a list of all pending events, sorted in the order of their time of appearance. The event that is to occur next is the first one in this list. Each event is represented by a list element (2) as in fig 1. The event queue is handled by an event



Next	Pointer to next list element.
ID	id-number (integer)
Time	Time of next occurance (double)
Period	Period time of repeated event (double)
Repeat	Boolean (true=repeated event)
Name	Pointer to NFA of word to be executed.
Ex.addr	Execution address.

Fig 1. Event queue element

scheduler which executes as a concurrent process. This may be acheived either by using 'genuine' (=time shared) concurrent processes (3) checking the real-time clock or by a real-time clock interrupt service routine. In the latter case we may use an interrupt signal from a hardware timer. Each time an event is activated the asociated word is executed. If this is a single non-repetitive event (AFTER), the corresponding list element is then discarded. In the case of a repetitve event, the a new time of occurence is calculated and the event is rescheduled by sorting the list element into the event queue. The reader is referred to (2) for the details of the list handling operations.

The AFTER and EVERY words create and sort list elements into the event queue in an obvious manner. The ID-number is an unique number for each event. The WHENEVER..PERFORM structure is implemented as a repetitive (EVERY) event where the condition is checked at each time of occurance. Should the result of the condition part be a true value, the action part is also executed.

4. Applications

The syntax described in this paper has been successfully been used in controlling heating systems. Typical installations use 2-10 heat pumps and several oil furnaces controlled by a single 6809 FORTH based computer, provide central heating and hot water for 20-50 appartments. The computer, designed specifically for industrial control applications, in these heating systems typically measures 10-20 temperatures and controls 10-30 relays and 4-8 valves. The FORTH event handling software consists mainly of roughly 20-50 events. Among these several PID-controllers are run as EVERY events.

5. References

1. Starling,M.K., "Of Widgits and Clock Ticks", Proc. 1984 Rochester FORTH Conference.

2. Olofsson,B., "FORTH List Handling", FD Vol 6, No. 1, May/June 1984

3. Zander,J., "FORTH Semaphores", FD Vol 6, No 4, November/December 1984