

RC3500 MONITOR 2

by

Ole Brun Madsen

&

Per Mondrup

RECAU-75-

19 pages

RC3500 MONITOR 2.

MONITOR 2	1
General Description	1. 1.
MONITOR PROCEDURES	2
SIGNAL	2. 1.
RETURN	2. 2.
WAIT	2. 2.
GETSHARE	2. 2.
STANDARD PROCEDURES	3
LOAD	3. 2
UNLOAD	3. 2
STOP	3. 3
START	3. 3
CREATE	3. 4
REMOVE	3. 5
DEFSHARE	3. 6
PROCESSES	4
SHARED VARIABLES	5
SEMAPHORES	6

MONITOR 2

General Description.

The MONITOR 2 in RC3500 is a collection of procedures responsible for the communication among processes and for the scheduling of processes.

The communication among processes is accomplished by means of queued semaphores and shared variables.

The monitor provides the facilities for execution of a set of processes in a manner that makes them appear as parallel processes.

As the main purpose with RC3500 is to act as a multi device controller, special attention has been paid to the run time scheduling to handle fast process shifts and interruption responses.

The MONITOR 2 is a redesign of MONITOR 1 [1]

The reasons for introducing a new monitor were to speed up the monitor functions and to be able to run programs in the high level language PLATON [2].

Definitions.

The following definitions are seen from the monitor's point of view.

Process: A process is an area of store containing the parameters (process descriptor) and runtime stack, necessary to execute a program.

Semaphore: A semaphore is a variable able to describe a queue of waiting processes or a queue of shared variables.

Shared variable A shared variable is two areas, the share head containing the parameters used to communicate and describe the contents of the share data.

[1]: RC3500 Monitor 1 RCSL 52-AA187
A/S Regnecentralen
3-9, Guldsmødegade
DK-8000 Aarhus C.

[2]: PLATON RECAU-75-

Monitor Procedures:

The monitor has two essential procedures seen from the point of view of the processes.

SIGNAL. The signal procedure is used to communicate a shared variable from a process to a semaphore.

The monitor performs an operation, that indivisibly unlinks the shared variable from the process descriptor and links it into a specified semaphore.

WAIT. The wait procedure is used to synchronize the execution of the process to the appearance of a shared variable at a specified semaphore.

The monitor performs a temporary stopping of the calling process until a shared variable is received at the semaphore. Furthermore the monitor indivisibly unlinks the shared variable from the semaphore and links it into the process descriptor.

Scheduling.

The monitor is responsible for the runtime scheduling.

When a process is activated the monitor decides from the process descriptor parameters how the process has to be scheduled.

A process can be scheduled to have the processor in a specified timeslice with a given frequency. At creation time the process is then placed in an active queue in accordance with its demands.

All such processes constitute a two dimensional system with the one dimension being the frequency and the other being the queue of active processes.

At runtime the monitor is interrupted by the clock system with a specified set of frequencies. The monitor then stops the current process and starts scanning the active queues in accordance with the specified frequency to find a runnable process.

If there is no runnable process a dummy process is activated. (idle loop).

MONITOR PROCEDURES:

General description.

Monitor procedures are represented as a sort of standard procedures.

Monitor procedures are used to secure indivisible operations that can not be handled by the processes directly.

In PLATON the monitor procedures are not declared.

In general a monitor procedure is called as:

<ident>(<parameterlist>);

loader information : → [83] → <data> →

where <data> is the internal number of the standard procedure.

Internal representation:

ist WI S XI <interrupt level>

The loader will place this instruction in the current load address and increase current load address with 2 bytes.

Execution:

Before call the parameter must be placed in W1.

After call the return information is placed in W1.

SIGNAL p TO s:

The shared variable p is signaled to the semaphore s.

p is unlinked from the shareque of the process, and not accessible from this process any longer.

loader information [83] data=0

execution:

before call:

p . destsem :=s

p . backward chain :=predecessor to p.

WI :=p.

RETURN p;

The shared variable referenced by p is signaled to the origin semaphore of the shared variable.

This procedure is essential the same as the monitor procedure SIGNAL.

loader information: [83] <data = 2>

execution:

before call: p . destsem := p . origsem.
 p . backward chain := predecessor to p.
 W1 := p.

WAIT p FROM s;

Synchronizes the process to the access of the shared variable referenced by p through the semaphore s.

The process will not proceed to execute the program text before a shared variable is received.

loader information: [83] <data = 4>

execution:

before call: W1 := S
 after call: W1 = p
 p . chain = W3 . share chain
 W3 . share chain = p.

ALLOC p FROM c {WITH s}

This procedure is used to get access to a shared variable declared internal to the process, and to assign a origin semaphore s to

The procedure is essentially the same as wait.

loader information: [83] <data = 6>

execution:

before call: W1 := anonymous semaphore of p.
 after call: W1 = p
 p . chain = W3 . share chain
 W3 . share chain = p
 p . origin semaphore := s.

STANDARD PROCEDURE S.

General Description.

Standard Procedure are used to execute operations on processes that is not possible with standard PLATON statements and from a security point of view not allowed to the user.

In PLATON the Standard Procedures are not declared. In general a Standard Procedure is called as:

<ident> (<paramlist>);

Loader information:

→ [83] → <data> →

where <data> is the internal number of the standard procedure.

Internal representation:

```

      ldr C W1 I      ;
      <address>      ;
      xsr W0 I X1     ;
  
```

The loader will place this sequence from current load address an 6 bytes foreward. Current load address is increased by 6 bytes.

Execution:

Before call the parameters must be placed in the stack and the work area as specified for each standard procedure.

After call the return information is placed in the stack registers and the work area as specified for each standard procedure.

In case of an runtime error in connection with the standard procedure call the error condition is returned in register W1 with the following convention:

```

      W1 = 0      ; No error
      W1 <> 0     ; Error number.
  
```

LOAD son FROM dev REPORT e;

LOAD is used to load a program text from input and link it to the process variable son.

Load checks that a program text is not already loaded.

loader information: [83] <data = 8>

execution:

before call: work1:= son : (process)
work2:= input : (word)

after call: son. current program := loaded program text.
son. process counter := 0
son. initial program := unchanged

errors:

W1	interpretation
0	no errors
2	program already loaded
4	loader error
6	ressource error

UNLOAD son REPORT error;

UNLOAD is used to remove a programtext. UNLOAD checks that there is no other processes connected to the program text.

loader information: [83] <data = 10>

execution:

before call: work1 := son (process)

after call: son. current program := NIL
son. process counter := 0
son. initial program := unchanged

errors:

W1	interpretation
0	no errors
2	no programtext
4	other users of the program text.

STOP x REPORT error;

STOP is used to stop a process represented by the shadow variable X and all subprocesses to x. Stop means that the process are not allowed to execute its program text before the process is started again.

When STOP is called on x, STOP is first called on x, so that all the subprocesses are stopped from the leaves towards the root.

loader information: [83] <data = 18>

execution:

before call: work1 := shadow ^{addr}~~value~~ of x

errors:

W1	interpretation
0	no errors
2	no process X
4	the process already stopped.

START x REPORT error;

START is used to start a process represented by the shadow variable x and all subprocesses that has been stopped by previous call of STOP on x.

Start means that the process is allowed to execute its program text.

loader information: [83] <data = 16>

execution:

before call : WORK1 := shadow ^{addr}~~value~~ of x

errors:

W1	interpretation
0	no error
2	no process x
4	x already started.

CREATE x LIKE son (parameters) WITH stacksize REPORT error;

CREATE is used to create a process to be represented by the shadow variable x over the programtext son with the parameters as specified in the son - parameter list.

CREATE essentially builds up a process descriptor and a stack with the size stacksize, and transfers the specified parameters to this stack.

loader information: [83] <data = 12>

execution:

before call:

Work1 := stack address (STA)
~~Work2~~ := ~~Shadow addr~~ X
 STA + 0 := stack size

STA + 2 := son address

STA + 4 := length of parameterlist to be transfered (N)

STA + 6 := first parameter

.....

STA + 4 + N := last parameter

after call:

~~X~~
~~work1~~ := shadow ~~value~~ ~~x~~ ^{addr}

son. processcounter := son. processcounter + 1;

errors:

W1	interpretation
0	no errors
2	no programtext
4	no space for stack.

REMOVE x REPORT error;

REMOVE is used to remove a process represented by the shadow variable x, that is to return all resources from x and all subprocesses of x. REMOVE is called on all sons of x before x is removed.

To secure that the removing of resources can be done without any harm, STOP is called before the actual remove procedure.

loader information: [83] <data = 14>

execution:

before call: Work1 = shadow ^{addr}value x

after call: son. process counter := son. process counter - 1

errors:

W1	interpretation
0	no errors
2	no process to remove.

defshare (error, semaphore, size, number):

Used to require a set of share variable with a given size, and queue them up at the semaphore specified.

The procedure is not allowed in a PLATON program, and is only used in process initialization.

loader information: [83] <data = 20>

execution:

WORK0 = semaphore
WORK1 = size
WORK2 = number

errors:

W1	interpretation
0	no errors
2	no resources

PROCESSES

GENERAL DESCRIPTION

A process is an area of store containing the parameters (process descriptor) and runtime stack, necessary to execute a program.

The runtime stack is only used by the process itself, that is to contain the variables of the program.

The process descriptor is used by monitor and the the standard procedures. The process has only a limited access to this area at runtime.

In PLATON the process concept is represented by two types of variables, namely a process variable used to represent the program text and a shadow variable used to represent the runtime stack and process descriptor.

A process can declare a set of process and shadow variables. These variables can only be handled by standard procedures at runtime.

As all PLATON programs are compiled into reentrant code many shadow variables can be created to the same process variable (program text).

Internal repretation:

process variable: STA + 0 : pointer to actual programtext.

STA + 2 : process counter.

STA + 4 : pointer to initial programtext.

where STA means stack, and process counter represents the number of processes actually executing the actual program text.

By means of the standard procedures load and unload the program variable can be altered at runtime by the process declaring it.

Shadow variable: STA + 0 : pointer to process descriptor.

By means of the standard procedures create, remove, start and stop the

shadow variables can be operated on at runtime by the process which has declared them .

Notice. A process can not operate in this manner with its own process variable and shadow variable.

Consequently the root process in the process tree has to be established separately in the system.

Process variables:

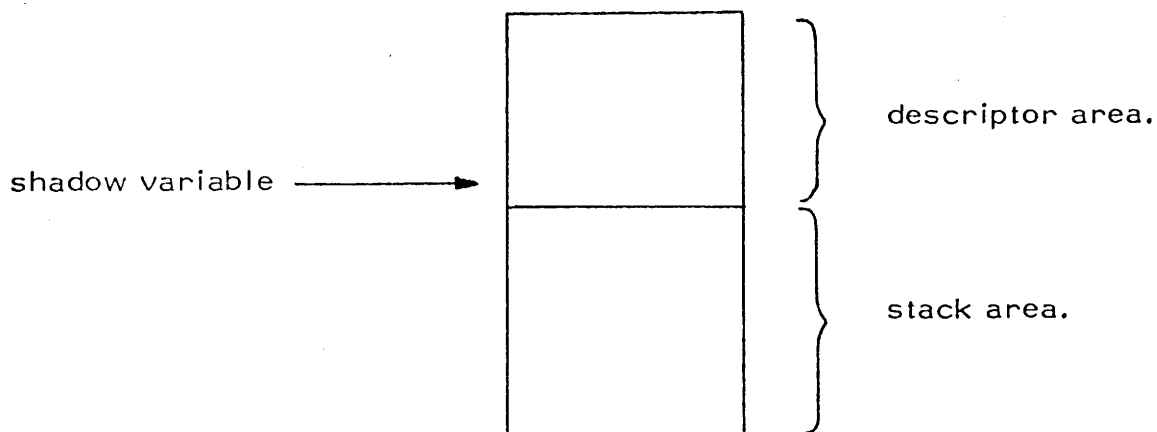
A process variable is represented in segments. If a process declaration contains nested process declarations each is allocated on a separate set of segments.

A process can only operate on the outermost level of subprocesses.

Process descriptor and runtime stack.

The process descriptor and runtime stack is allocated as two contiguous storage areas. The shadow variable is pointing to the top of the descriptor.

The descriptor is addressed with negative addresses relative to the shadow variable, and the stack with positive addresses relative to the shadow variable:



When a process is executed the contents of the shadow variable must be hold and unchanged in register w3.

The process descriptor contains the necessary variables to describe the process. It represents the resources of the process, the dump area for the registers when a process is stopped, entry and exit points to the program text, status, etc.

The monitor is using the following variables to handle a process:

process chain: pointer variable to link the process into a semaphore
[w3 - 1] used by the runtime system to determine whether a process is waiting or active.

activequeue chain: pointer variable used to link the process into the system of
[w3 - 3] active processes.

process status:
[w3 - 5] variable used by the runtime system to determine if a process is started.

activation procedure:
[w3 - 7] variable used to describe the monitor procedure used to activate the process.

sw0, sw1, sw2: Three variables used to dump the registers when a process is
[w3-9, w3-13] temporary stopped by the runtime system.

first brother, first son:
[w3-15, w3-17] two links used to represent the runtime process tree.

share queue forward:
[w3 - 19] variable used to point to the list of shared variables recived by the monitor procedure wait.

Futhermore the system and runtime system makes use of the following variables:

shared queue backward:

[w3 - 21] variable used to maintain a backward chain through the above mentioned shared variables.

system semaphore:

[w3 - 23] semaphore address used by the system to redefine the origin semaphore of a local shared variable when the process is removed.

init. address:

[W3 - 25] Variable used by the system to hold the initial entry point to the process.

exit address:

[W3 - 27] Variable used by the system to hold the exit point to a process. When a process is removed by the system, the system forces the process to the exit address, to secure that all its subprocesses and other resources are removed and returned. A process can at any point in execution be forced to exit.

status:

[W3 - 29] Variable used by the system to hold the necessary execution status, i. e. whether the process is executing its initial code, its program text or its exit code.

maximum stack code:

[W3 - 31] Variable used by the system to define the maximum stack assigned to the process.

dump W2 stackpointer:

[W3 - 33] Variable used by the Platon programs for temporary storage of the current stack pointer.

work 0 - 4:

[W3 - 35, W3 - 43] Variable area used by the system to exchange parameters
(standard procedures etc.)

programtext:

[W3 - 45] Variable used by the system to connect the variable defining
the current programtext and the process.

sys w0 - sys w2
[w3-47, w3-51]

Variables used by system std. procedures for
reg. dump

father
[w3-53]

Pointer to father process

SHARED VARIABLE S.

General Description.

A shared variable is two areas of store used to exchange information between processes.

In the present implementation only one process can access the contents of a shared variable in a given moment.

The two areas of a shared variable is the shared data and the shared head.

The shared data is used to contain the information to be communicated.

The shared head is used to describe the shared data area, and to held the necessary information for the system to identify the shared variable.

Internal representation:

<u>shared head:</u>	ADDR + 0:	foreward chain
	- 2:	backward chain
	- 4:	origin semaphore address
	- 6:	destination semaphore address
	- 8:	shared data start address
	- 10:	shared data end address
	- 12:	shared data actual size
	- 14:	shared data code.

foreward chain: used by the monitor to link the shared variable to a semaphore or to the process shared queue chain . This variable must never be changed by the processes.

backward chain: used by the monitor (jvf. p. 4. 1) to maintain the process shared queue. The monitor uses the backward chain in connection with unlinking of a shared variable.

origin semaphore address:

used by the system to define the ownership of a shared variable. This address must only be changed by the owner process.

When a process is removed the origin semaphores is redefined to system semaphore to secure that all its shared variable temporary circulating among other processes will be returned to the system, and not to the removed process.

destination semaphore address:

used to define where the shared variable has to be send.

This address may be set to any semaphore address known by the process.

shared data start address:shared data end address:

used to define the psysical shared data area.

This addresses must only be changed by the standard procedures.

shared data actual size:

used to define the logical size of the shared data area.

This variable is set, when a shared variable is requested. The value of the variable will always be less or equal to the physical size of the shared data area.

shared data code: variable that can be user defined, and used for identification.

SEMAPHORES.

General Description:

A semaphore is an area of store containing a queue head and a state.

The queue head can either describe a queue of waiting processes or a queue of signaled shared variables. The queue may be empty.

The state of a semaphore represents the type of the queue (processes or shared variables).

A semaphore is used to synchronize the access to a shared variable from a process. The only way a process can get access to a shared variable is to execute the monitor procedure WAIT.

Implementation.

<u>semaphore</u>	ADDR + 0	pointer to first queue element
	ADDR - 2	pointer to last queue element

The representation of the state is implicit. A pointer to a process is always odd and a pointer to a shared variable is always even. A passive semaphore (A semaphore pointing to an empty queue) is represented with the two pointers pointing to the semaphore address.

The elements in the queue are linked together, with the last element pointing to the semaphore.

Shared variables are linked together through the share head forward chain. in the sequential order they are signaled to the process.

Processes are linked together through the process descriptor process chain.

Defined semaphores must only be operated by the standard procedures or the monotor.