00's

Title:

THE COROUTINE - MONITOR FOR RC 3600

A/S REGNECENTRALEN

RC SYSTEM LIBRARY: FALKONERALLE 1 DK-2000 COPENHAGEN F

RCSL No: 43-R10371
Edition: July 1976
Author: Jørgen Lindballe

Keywords:

RC 3600, Coroutine-Monitor, Coroutines, Semaphores, Program Description

Abstract:

The Coroutine-Monitor is a set of procedures for the RC 3600, the purpose of which are to ensure synchronization and exchange of data between coroutines in a process in a MUS-Multiprogramming System. 30 pages.

Printed by A/S Regnecentralen, Copenhagen

THE COROUTINE-MONITOR FOR RC 3600

CHAPTER 1:

COROUTINES AND PROCESSES

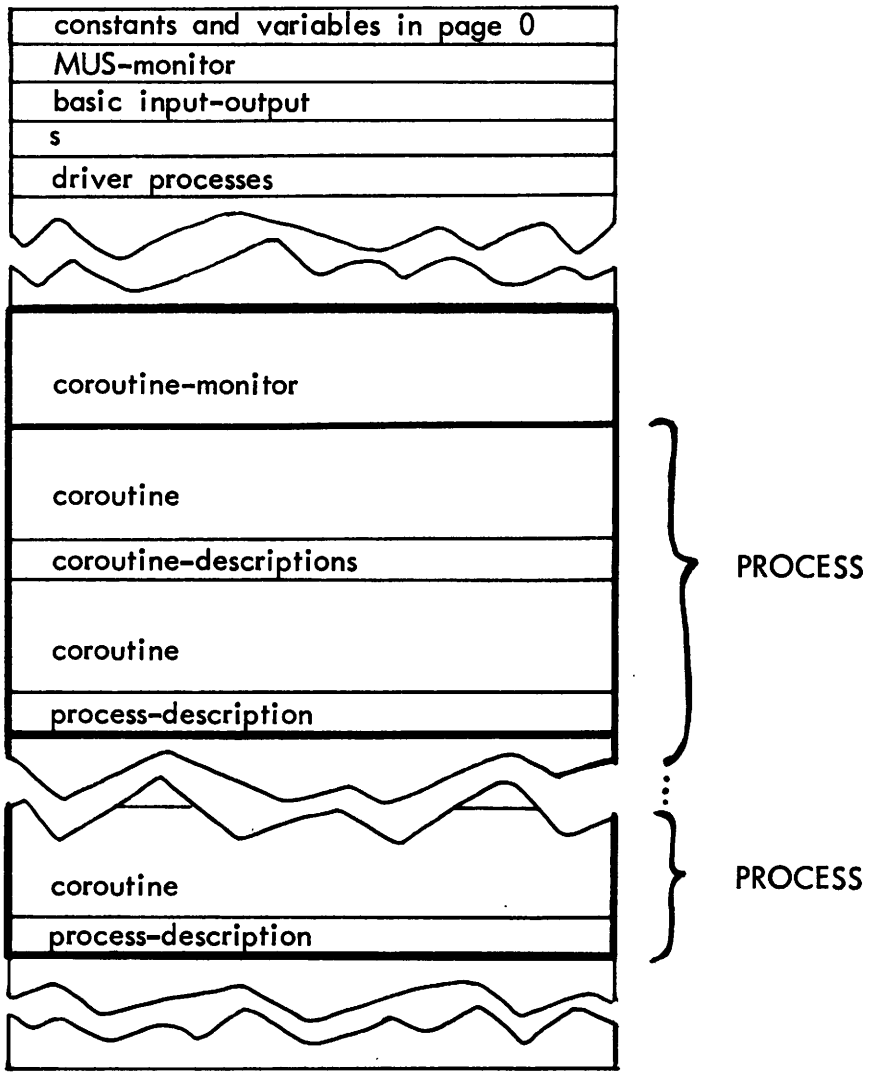


Fig. 1.1: RC 3600 MEMORY LAY-OUT

Fig. 1.1 illustrates a typical RC 3600 memory lay-out. Besides the basic software the memory contains a number of processes. Each process is terminated by its process-description. The memory contains the Coroutine-Monitor, too. One or more of the processes consist of a number of coroutines and the descriptions of the coroutines.

So the memory contains a normal MUS-system with two extensions:

a. The Coroutine Monitor has beed added.

b. Some of the processes are structured in socalled coroutines.

A coroutine is a piece of code, which is executed again and again in a cyclic manner only suspended in one or more waiting points. The execution of code between two waitingpoints is done in an indivisible way, i.e., the coroutine is not interrupted by other coroutines in the process. The coroutines in a process are synchronized and they exchange data by means of socalled semaphores (ch. 2.4).

So a coroutine is either active, or it is waiting in one of the queues, of which the most important are the semaphore-queues (in which the coroutines are waiting for signals to the semaphores) and the active queue (coroutines ready for the execution of instructions, because signals has been received).

But the processes still exchange data by means of the MUS-procedures (send message - wait answer, etc) and they share the CPU-time in the ordinary way.

CHAPTER 2:

DESCRIPTIONS, SEMAPHORES AND OPERATIONS

2.1. DESCRIPTIONS

Fig. 2.1 shows as an example how the different coroutine-descriptions (representing the coroutines) for one process are linked into different queues.

The current process are pointed out by "CURRENT PROCESS". The active coroutine and the active-, the answer-, and the delay-queues are linked to pointers in the process description. The semaphore queues and the operation queues are linked to semaphores.

The Coroutine Monitor allows a coroutine to wait not only for an answer but also for a message (ch. 4.3).

2.2. PROCESS DESCRIPTIONS

Every process is terminated by a process-description used by the MUS-Monitor. Used together with the Coroutine Monitor the process description must be extended by the following words:

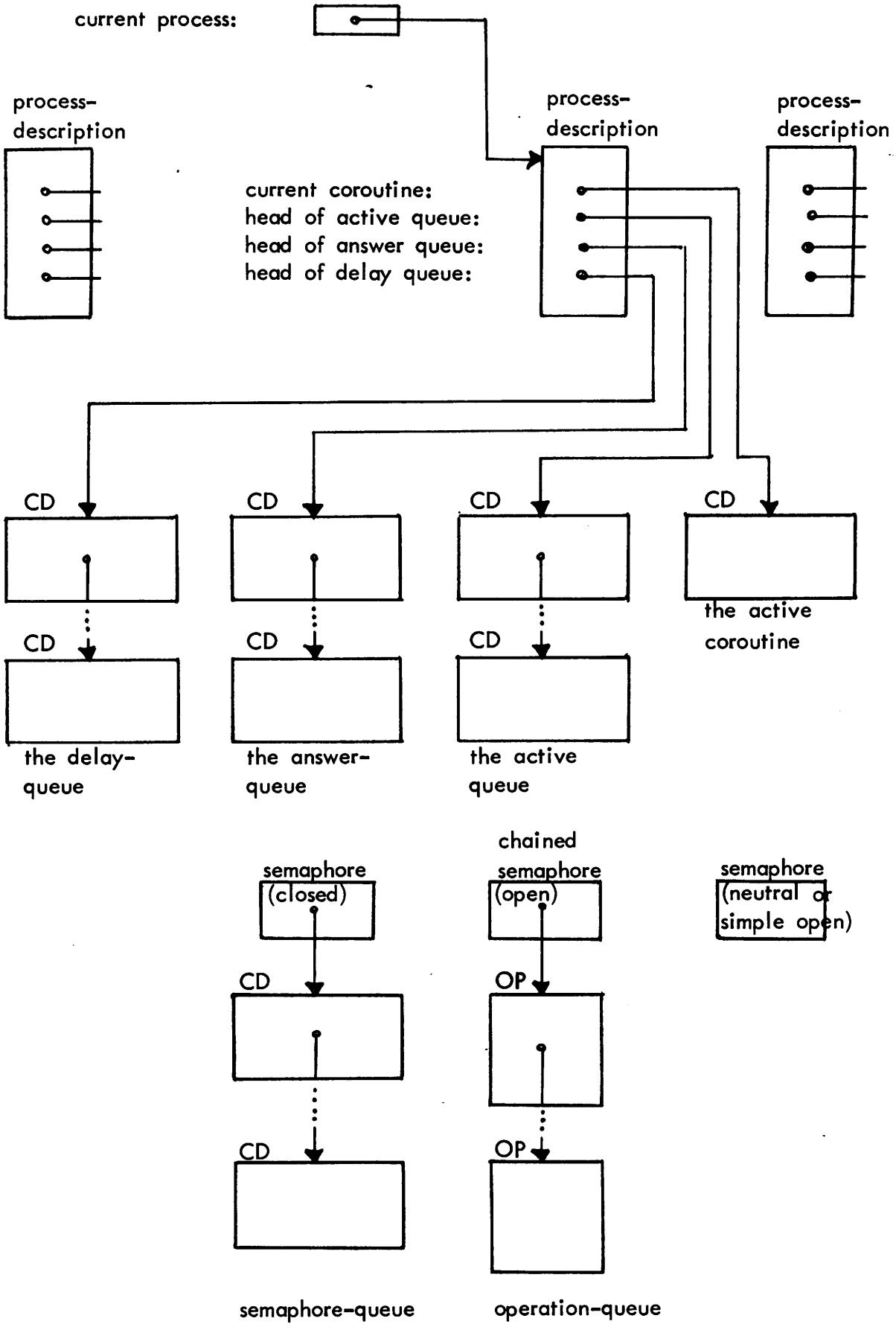


Fig. 2.1: QUEUED COROUTINE-DESCRIPTIONS AND OPERATIONS

COROUTINE-ELEMENTS IN PROCES-DESCRIPTION:

Name:	Contents:
CCOROUT	Address of the description of the current coroutine (the active coroutine).
LATIME	Clock value of the latest time the delay queue was scanned by the coroutine monitor.
HACTIV	Address of the description of the first coroutine (the head) of the active queue.
HANSWER	Address of the description of the first coroutine (the head) of the answer queue.
HDELAY	Address of the description of the first coroutine (the head) of the delay queue.
(TRETURN	a: Return address from CTOUT b: Message buffer address).
(TRECORD	Address of test record).

The two last words are only used by the test-procedures (chapter 8).

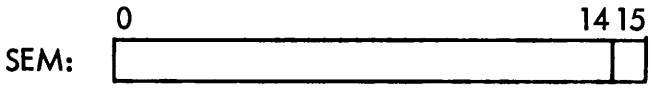
2.3. COROUTINE DESCRIPTIONS

Each coroutine has its own description, which may be stored everywhere inside the process. A coroutine-description consists of the following 6 words:

Name:	Contents:
CIDENT	CIDENT (0) : Priority (ch. 4.2) CIDENT (1:7) : Test (for test only) CIDENT (8:15) : Ident (for test only) (ch. 8.1)
NEXT	0 (termination) or address of next coroutine description in queue.
CEXIT	Return address to coroutine after a waitingpoint. (ch. 3)
CLATOP	Latest operation (ch. 6).
CRETURN	Return address of coroutine after call of procedure.
CAC1SAVE	Saved AC 1.

2.4. SEMAPHORES AND OPERATIONS

A semaphore is a word which may be referred to by a name from every co-routine inside the process:



The contents is changed by the coroutine-monitor-procedures. It is used for synchronization of the coroutines and for exchange of data (operations) between the coroutines.

A semaphore may be either a simple semaphore or a chained semaphore.

A semaphore may be open, closed or neutral.

The contents of a semaphore depends on whether it is simple or chained, and on whether it is open, closed or neutral according to fig. 2.2. The head of the semaphore-queue is the description of the first coroutine in the queue. The head of the operation-queue is the first operation in the queue.

An operation is a record of data in which word no. 0 may be used for the link to the next operation (or the termination of the operation-queue).

state \ kind	simple	chained
open	The count is inereased by 1 for each call of SIGNAL (and decreased by 1 for each call of WAIT). SEM(0:14):COUNT (>0) SEM(15):1	One or more operations are linked to the semaphore after SIGNAL-CHAINED SEM(0:14):Addr. of head of op.queue SEM(15):1
neutral	The semaphore may be opened by SIGNAL or closed by WAIT. SEM(0:14):0 SEM(15):1	The semaphore may be opened by SIGNAL-CHAINED or closed by WAIT-CHAINED SEM(0:14):0 SEM(15):1
closed	One or more coroutines are linked to the semaphore after call of WAIT SEM(0:14):Addr. of head of sem.que SEM(15):0	One or more coroutines are linked to the semaphore after call of WAIT-CHAINED SEM(0:14):Addr. of head of sem.que SEM(15):0

Fig. 2.2

2.5. CONSTANTS AND VARIABLES IN PAGE 0

Below are listed the locations in page 0 which are used by the coroutine-monitor:

Name:	Contents:

COROUT	Address of the description of the current coroutine (active coroutine) in current process. (Contents equal the contents of CCOROUT.CUR).
	COROUT may be omitted; it has been introduced to save one instruction:
	LDA 2, COROUT
	instead of LDA 2, CUR
	LDA 2, CCOROUT, 2
	and it has complicated the MUS-Monitor which must update COROUT, when changing from one process to another. Furthermore COROUT must be initialized from the coroutines.

PC	The program counter is used by COROUTINE-WAIT-ANSWER.
CUR	Address of the description of the current process.
RTIME (1:2)	Real Time Clock.

EP of Coroutine-Delay:	CDELAY
EP of Wait:	WAITSEM
EP of Wait-Chained:	WAITCHAINED
EP of Coroutine-Wait-Asnwer:	CWANSWER
EP of Ctest:	CTEST, ch. 8
EP of Cprint:	CPRINT, ch. 8
EP of Ctout:	CTOUT, ch. 8
EP of Signal:	SIGNAL
EP of Signal-Chained:	SIGCHAINED
EP of Coroutine-Pass:	CPASS

EP ≡ entry point

CHAPTER 3:

INITIALIZATION

Within a process one coroutine is born as the active coroutine, while the remaining coroutine-descriptions usually are linked together in the active-queue. In these descriptions the words CEXIT point at the initialization-code of the coroutines.

In each coroutine the initialization-code is terminated by a call of WAIT, WAIT-CHAINED or COROUTINE-WAIT-ANSWER, so that at last every coroutine is linked to the answer-queue or a semaphore-queue, and the process is waiting for an answer from another process.

The description of the process is born with CCOROUT pointing at the description of the abovementioned active coroutine; the "program counter" must point at the initialization-code of this coroutine. HACTIV is pointing at the first description in the active-queue mentioned above.

A semaphore may be born in a state one find appropriate.

PROCES DESCRIPTION:

CHAPTER 4:

COROUTINES AND QUEUES

4.1. THE ACTIVE COROUTINE

The socalled active coroutine (or current coroutine) is that coroutine within a process, which at the moment (more precisely: When the process is "current process") executes instructions.

This coroutine remains being the active one until it arrives a waitingpoint, i.e., it calls one of the following coroutine-monitor-procedures: WAIT or WAIT-CHAINED (and the semaphore is not open), COROUTINE-WAIT-ANSWER, COROUTINE-DELAY or COROUTINE-PASS; then it is linked upon one of the queues: Semaphore-, answer-, delay-, or active-queue, respectively, and the first coroutine in the active queue is linked off now being the active coroutine.

If the active-queue is empty, when the active coroutine arrives its waitingpoint, the Coroutine-Monitor will let the process wait for either the arrival of an answer for a coroutine in the answer-queue (this coroutine is then linked off being the active one) or the run-out of the delay for one or more coroutines in the delay-queue. These coroutines are then put into the active-queue.

The execution of instructions between two waitingpoints is done in an indivisible way in the sense that it is not interrupted by other coroutines within the process. But it should be emphasized that interrupt is not disabled, because the process must share the time with other processes. But interrupt is mainly disabled in the Coroutine-Monitor-procedures, because this monitor as the MUS-monitor is common to all processes.

In the Coroutine-Administration (chapter 5.10) which must be reentrant for all processes due to the call of WAIT, interrupt is, however, enabled.

4.2. THE ACTIVE-QUEUE

The active-queue is a queue of descriptions of the coroutines which are ready for the execution of instructions.

When an active coroutine arrives a waitingpoint, the first coroutine-description is linked off the active-queue being the active one.

Below are listed a number of situations, where a coroutine-description is linked on the active-queue. However, it must be pointed out that if the priority of the coroutine is high, it becomes the first one, and if the priority is low it becomes the last one in the queue:

a. When a semaphore is signalled from another coroutine the first coroutine is linked off the semaphore-queue.

b. When an answer arrives, the coroutine waiting for the answer is linked off the answer-queue. In this situation the active-queue is always empty, so the coroutine becomes the active one.

c. A coroutine waiting for the run-out of a delay is linked off the delay-queue, when this happens.

d. When the active coroutine calls COROUTINE-PASS. If the priority is high, however, the call is dummy, because the coroutine remains the active one.

4.3. THE ANSWER-QUEUE

The answer-queue is a queue of descriptions of coroutines each waiting for an answer described by the buffer address.

After having called SEND-MESSAGE it is put into (the front of) the answer-queue, when the coroutine calls COROUTINE-WAIT-ANSWER. Returning to the coroutine this must call WAIT-ANSWER to get the answer, i.e., the coroutine is made active, when the answer arrives.

It should be emphasized that the Coroutine-monitor calls WAIT only when the active-queue is empty. Contrary all coroutines should be delayed.

As WAIT-ANSWER may be called behind your back for example from the BASIC I/O (wait transfer) such procedures, when called from a coroutine, first call COROUTINE-WAIT-ANSWER.

Also COROUTINE-WAIT-MESSAGE is implemented by calling COROUTINE-WAIT-ANSWER (bufferaddr. = 0). But messages must only be received in one coroutine, because messages are not identified, contrary an answer. (Cfr. ch. 5.10).

4.4. THE DELAY-QUEUE

The delay-queue is a queue of descriptions of the coroutines each waiting for a delay. A description is put into (the front of) the delay-queue by the call of COROUTINE-DELAY. It is linked off the delay-queue and on the active-queue, when the time specified has gone.

The unit of the delay-interval is no. of (20 msec-) clock counts.

4.5. THE SEMAPHORE-QUEUES

A semaphore-queue is a queue of coroutine-descriptions linked on a (closed) semaphore. They were linked on the queue, when WAIT or WAIT-CHAINED was called and the semaphore was not open. A coroutine-description is always linked behind the queue independently of the priority. The first description is linked off and on the active queue, when another coroutine signals the semaphore by SIGNAL or SIGNAL-CHAINED, respectively.

4.6. THE OPERATION-QUEUES

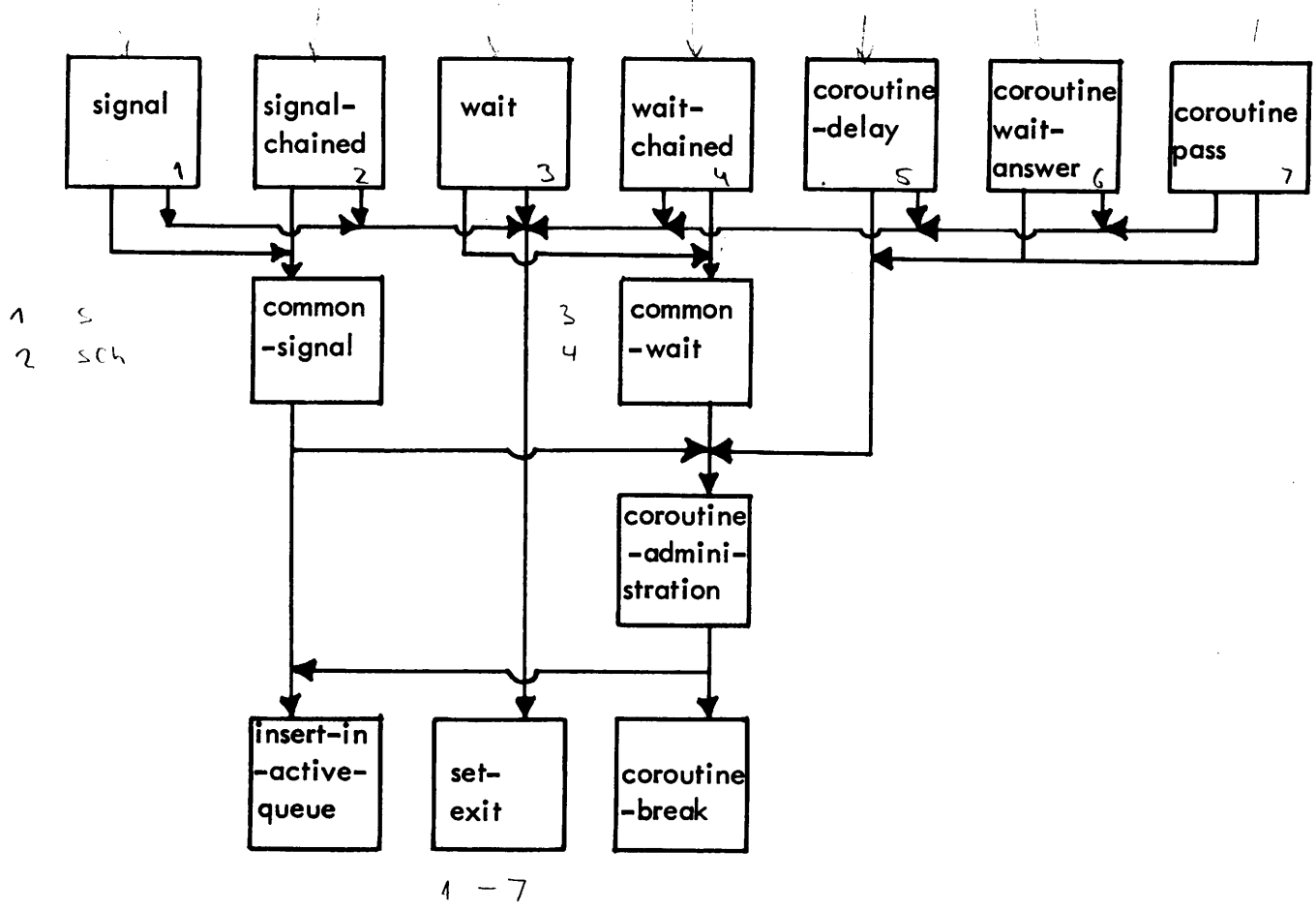
An operation-queue is a queue of operations linked on an (open, chained) semaphore.

It was linked on the queue, when a coroutine called SIGNAL-CHAINED, and it is linked off when a coroutine calls WAIT-CHAINED.

CHAPTER 5:

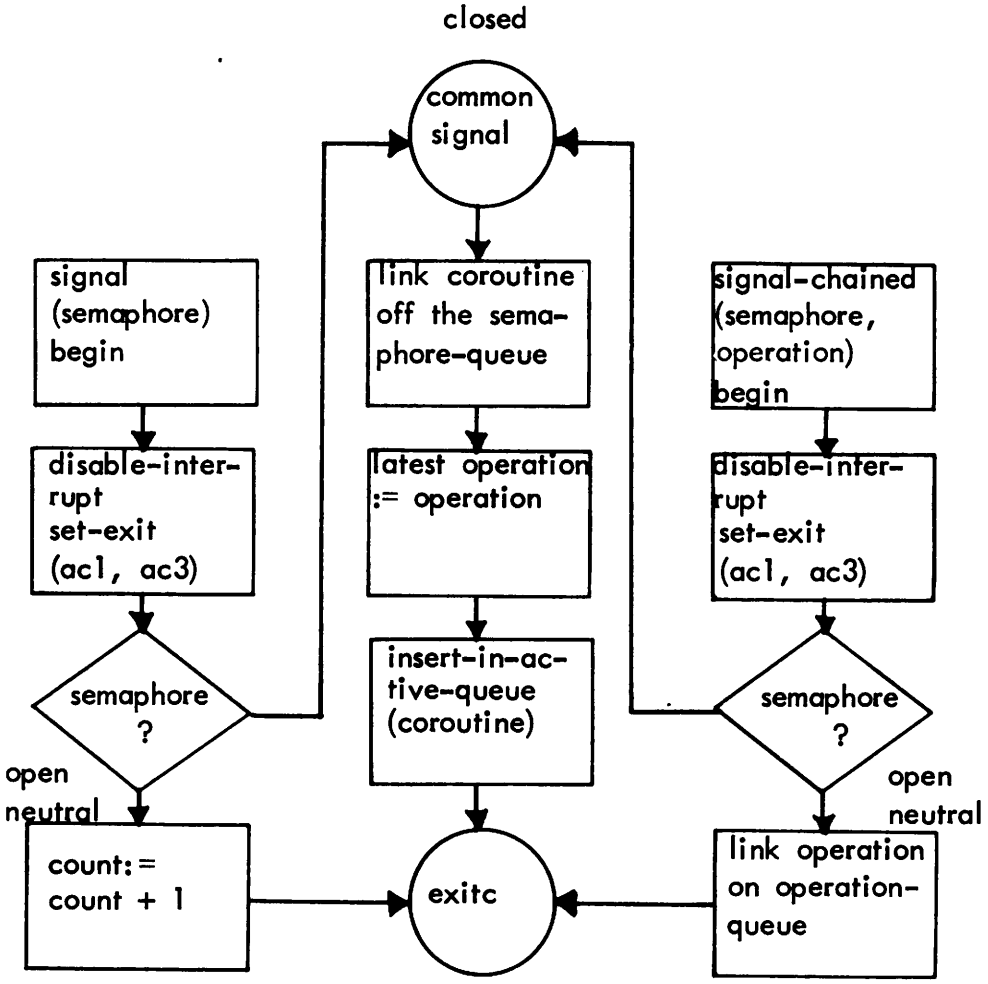
SYSTEM DIAGRAM AND FLOWCHARTS

5.1. THE COROUTINE-MONITOR

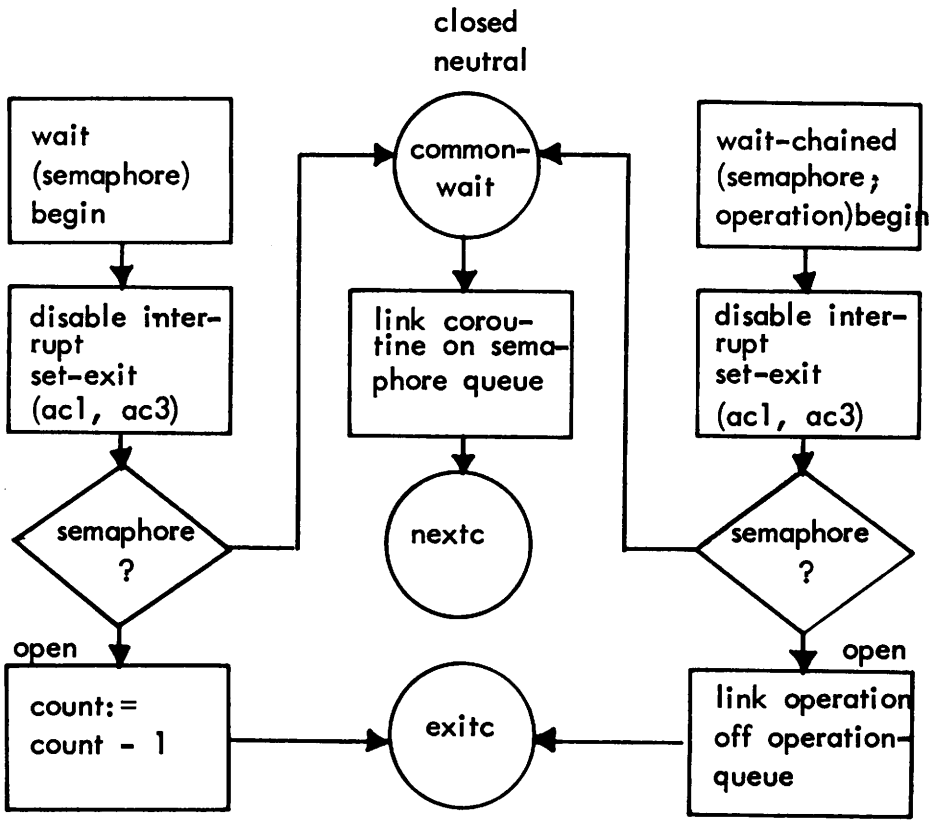


This system-diagram shows the hierarchical order of the procedures in the Coroutine-Monitor. In the upper row are shown the 7 procedures which may be called from a coroutine. The arrows indicate from where the remaining procedures are called.

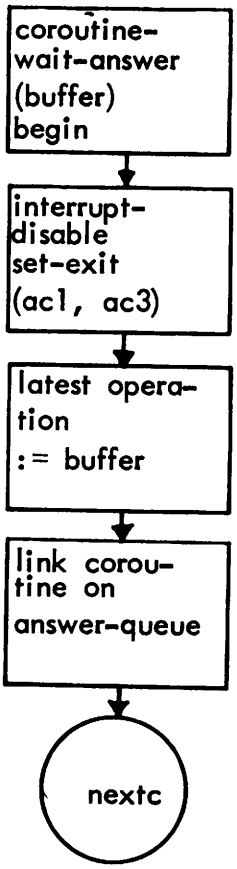
5.2. SIGNAL AND SIGNAL-CHAINED



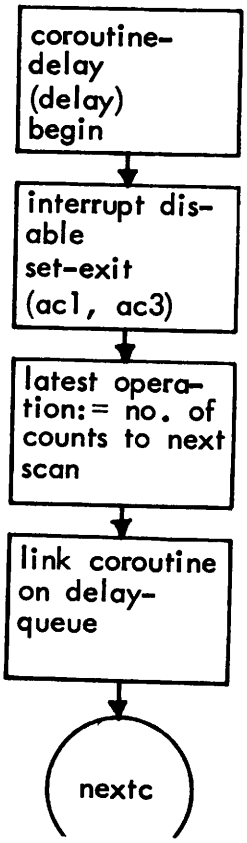
5.3. WAIT AND WAIT-CHAINED



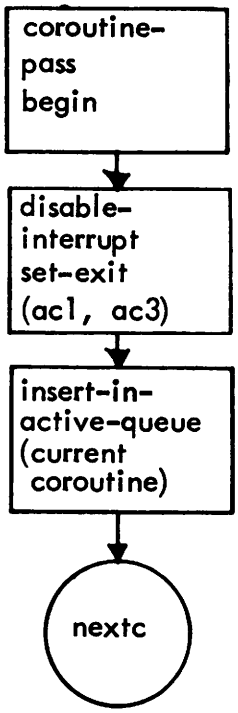
5.4. COROUTINE-WAIT-ANSWER



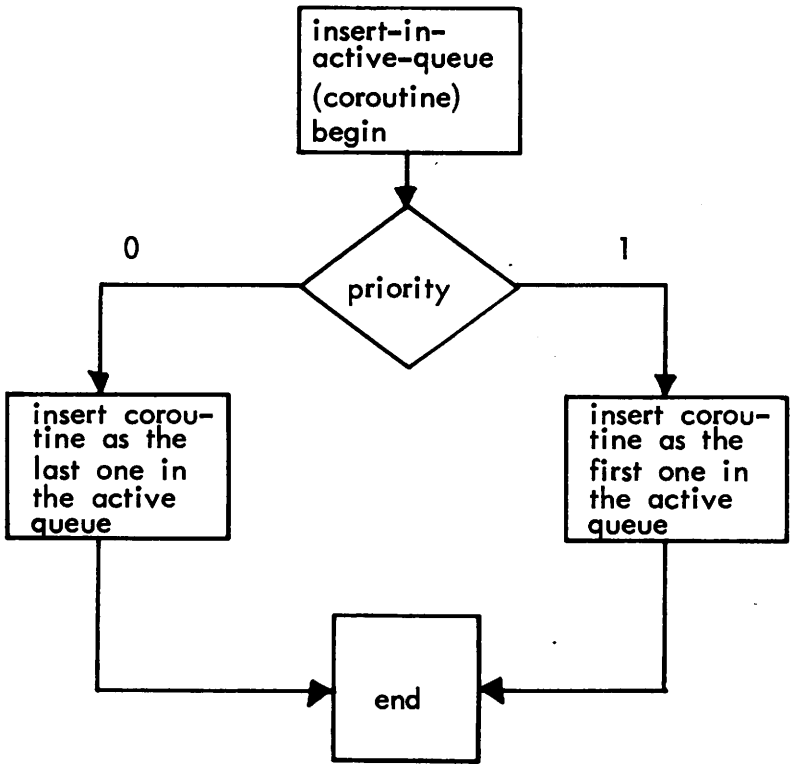
5.5. COROUTINE-DELAY



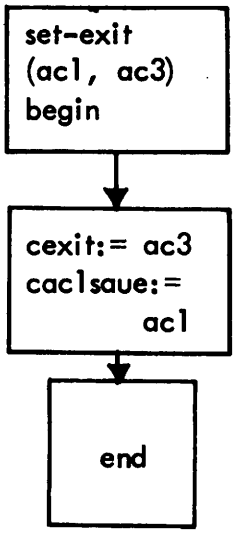
5.6. COROUTINE-PASS



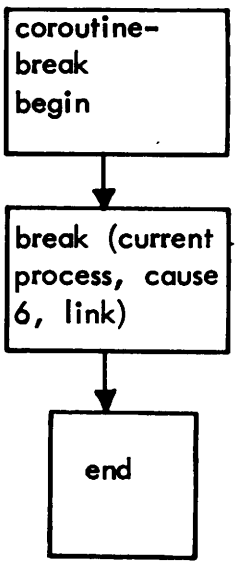
5.7. INSERT-IN-ACTIVE-QUEUE



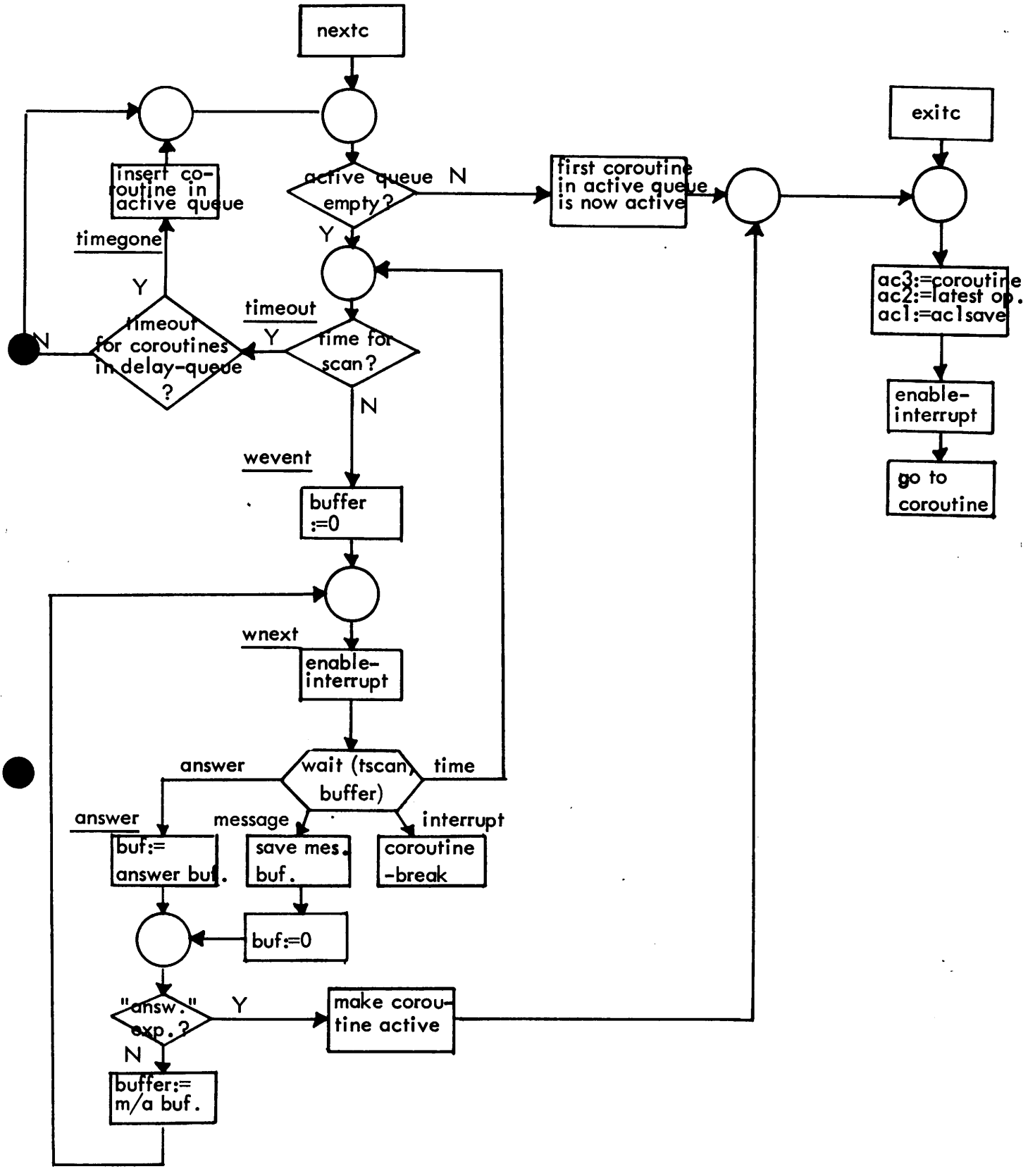
5.8. SET-EXIT



5.9. COROUTINE-BREAK



5.10. COROUTINE-ADMINISTRATION



CHAPTER 6:

CALL OF THE COROUTINE PROCEDURES

The assembler has been extended with the names of the procedures, so that they are called in the way mentioned below.

It should be emphasized that ac0 is always destroyed, that ac1 is never changed, that ac2 by return always contains CLATOP. COROUTINE, i.e., the address of the latest operation except for the call of COROUTINE-WAIT-ANSWER (ac2 = buffer address) and COROUTINE-DELAY (ac2 = no. of intervals), and that ac3 by return contains the address of the description of the coroutine itself (cfr. chapter 7).

6.1. PROCEDURE SIGNAL (SEMAPHORE)

	call:	return:
ac0		undefined
ac1	addr. of semaphore	addr. of semaphore
ac2		addr. of latest operation
ac3	link	addr. of coroutine-description

If the semaphore is not closed, its value is increased by one. Otherwise the waiting coroutine is inserted into the active-queue.

6.2. PROCEDURE SIGCHAINED (SEMAPHORE, OPERATION)

	call:	return:
ac0		undefined
ac1	addr. of semaphore	addr. of semaphore
ac2	addr. of operation	addr. of latest operation
ac3	link	addr. of coroutine-description

If the semaphore is not closed, the operation is linked on its operation-queue. Otherwise the waiting coroutine is inserted into the active-queue.

6.3. PROCEDURE WAITSEM (SEMAPHORE)

	call:	return:
ac0		undefined
ac1	addr. of semaphore	addr. of semaphore
ac2		addr. of latest operation
ac3	link	addr. of coroutine-description

If the semaphore is open, its value is decreased by one. Otherwise the calling coroutine is linked on the semaphore-queue.

6.4. PROCEDURE WAITCHAINED (SEMAPHORE, OPERATION)

	call:	return:
ac0		undefined
ac1	addr. of semaphore	addr. of semaphore
ac2		operation
ac3	link	addr. of coroutine-description

If the semaphore is open, an operation is linked off its operative-queue. Otherwise the calling coroutine is linked on the semaphore-queue.

6.5. PROCEDURE CWANSWER (BUFFER)

	call:	return:
ac0		undefined
ac1		unchanged
ac2	addr. of buffer	addr. of answer- or message-buffer
ac3	link	addr. of coroutine-description

The coroutine is linked on the answer-queue.
If by call ac2 = 0 the procedure waits for a message instead of an answer.

It should be noticed that after COROUTINE-WAIT-ANSWER you must call WAITANSWER.

6.6. PROCEDURE CDELAY (NO. OF INTERVALS)

	call:	return:
ac0		undefined
ac1	no. of intervals	no. of intervals of 20 mS.
ac2		undefined
ac3	link	addr. of coroutine-description

The coroutine is linked on the delay-queue.

6.7. PROCEDURE CPASS

	call:	return:
ac0		undefined
ac1		unchanged
ac2		addr. of latest operation
ac3	link	addr. of coroutine-description

The coroutine is again inserted in the active-queue.

CHAPTER 7:

REENTRANT COROUTINES

The main characteristics of reentrant code are a) variables are extracted from the code and b) variables are repeated as many times the code should be reentrant.

A n-time reentrant coroutine is consequently designed as shown in fig. 7.1, i.e., variables and coroutine-descriptions are repeated n times.

After return from a coroutine-monitor-procedure ac3 contains the address of the description of the coroutine itself. The reason is that in this way the addressing of the variables is more easily coded.

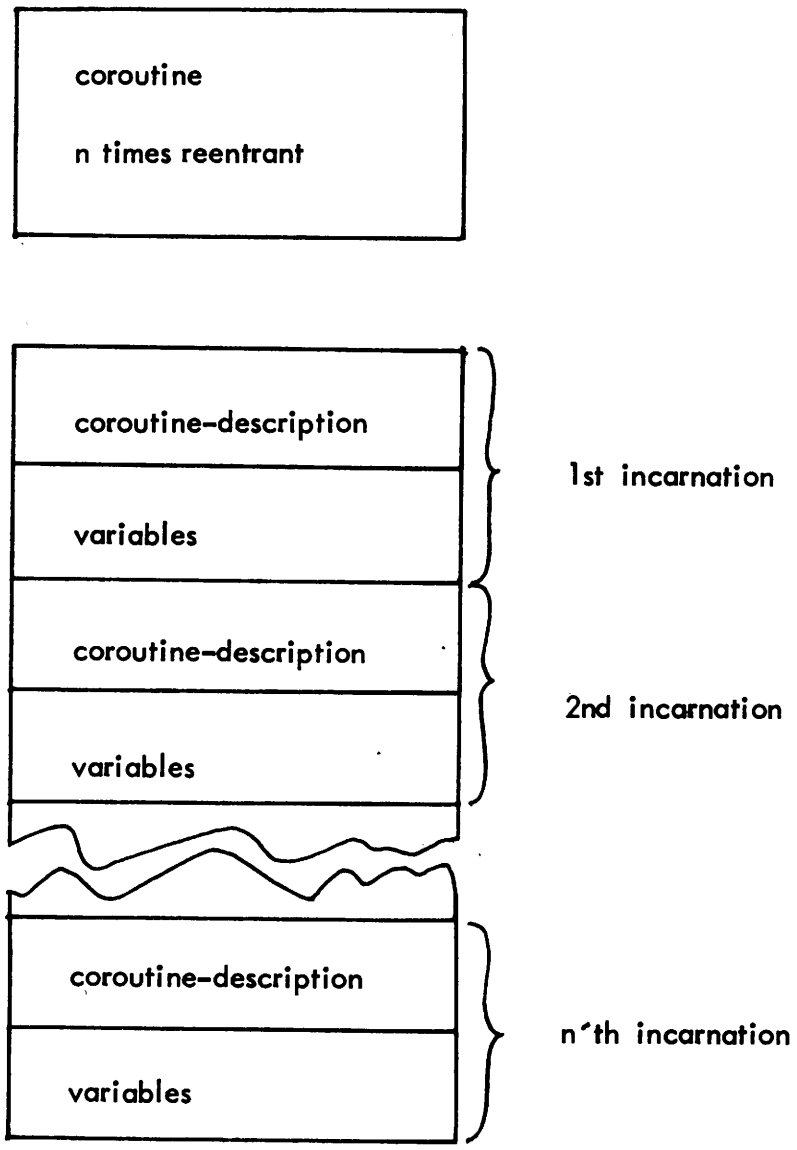


Fig. 7.1: REENTRANT COROUTINE, DESCRIPTIONS AND VARIABLES

CHAPTER 8:

THE TEST PROCESS

8.1. STRUCTURE

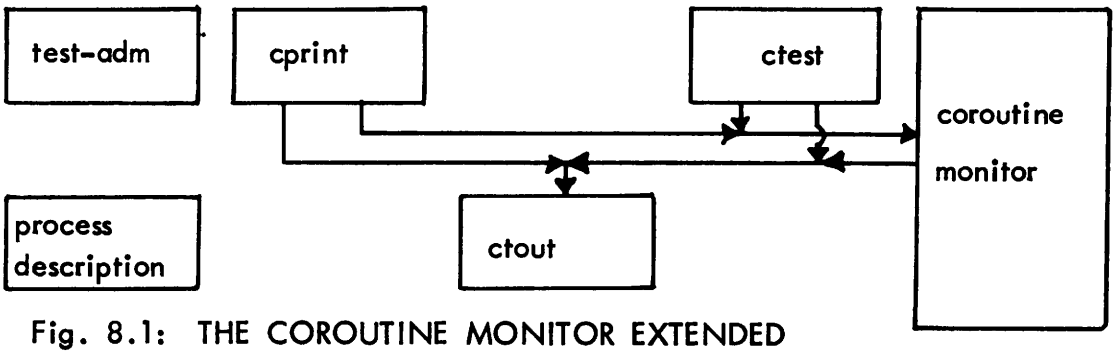


Fig. 8.1: THE COROUTINE MONITOR EXTENDED WITH THE TEST PROCESS AND THE TEST PROCEDURES

The Coroutine-Monitor may be extended with the Test-Procedures:

CPRINT
CTEST
CTOUT

and the Test-Process:

TEST-ADMINISTRATION
PROCESS-DESCRIPTION

The Test-Process and the Test-Procedures exist in two versions:
One version which dumps records in a memory area (TMEDIUM = 0), and another which dumps on magnetic tape (TMEDIUM <> 0).

Each Coroutine-Monitor-Procedure mentioned in chapter 6 calls CTOUT, which may be called, too, by calling CPRINT or CTEST. These two procedures call SET-EXIT.

Every time CTOUT is called, it generates a testrecord (fig. 8.2) if:

dataswitch (0) = 1 and
testoutput is wanted, when that coroutine calls that Coroutine-Monitor-Procedure, i.e. (cfr. ch. 2.3)

if CIDENT (1:7).CCOROUT ∧ INHIBIT.PROCEDURE = true

In other words: You may get testrecords when some coroutines call some procedures and when other coroutines call some other procedures etc.

By a message from CTOUT to TEST-ADM the testrecords are transferred to the zone for magnetic tape.

Address (in octal):		Name:	Contents:
Testrecord +	0	TKIND	long, inhibit, kind (from call)
	1	TPROC	process-description addr.
	2	TIDENT	ident of coroutine
	3	TTIME	time (0)
	4	TTIM1	time (1)
	5	TAC0	ac0 or ref (7) (cprint)
	6	TAC1	ac1 or ref (8) (cprint)
	7	TAC2	ac2 or ref (9) (cprint)
	10	TAC3	ac3 or ref (10) (cprint)
	11	TOPT1	ref (1)
	12	TOPT2	ref (2)
	13	TOPT3	ref (3)
	14	TOPT4	ref (4)
	15	TOPT5	ref (5)
	16	TOPT6	ref (6)

Fig. 8.2: TEST RECORD

When the testrecords are dumped in core, the TEST-ADM transmits the core-dump when it receives a message from another program (called TESTPRINT).

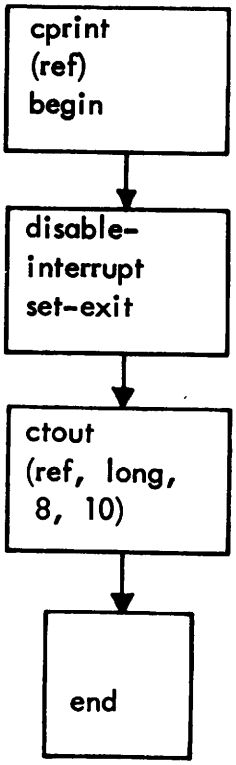
The parameter LONG is information to the TESTPRINT-program whether the 6 last words of the testrecord should be printed (LONG = true) or not (LONG = false). KIND is information about the procedure called.

It should be emphasized that the Coroutine-Monitor should not be used by more than one process, when testoutput is produced. The error is that interrupt is enabled in CTOUT, and it should be corrected by ensuring that CTOUT is not called directly from the coroutines.

PROCEDURE:	LONG:	INHIBIT:	KIND:
signal-chained	l	64	1
signal	s	32	2
wait-chained	s	16	3
wait	s	16	4
coroutine-wait-answer	s	1	5
coroutine-pass	s	2	6
exit	l	4	7
coroutine-delay	s	2	8
ctest	l	8	9
cprint	l	8	10

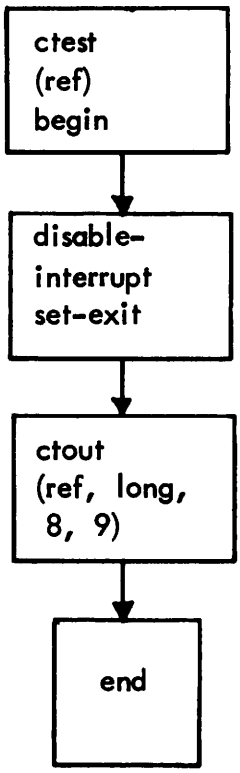
Fig. 8.3: LONG, INHIBIT AND KIND FOR THE COROUTINE-MONITOR-PROCEDURE

8.2. CPRINT



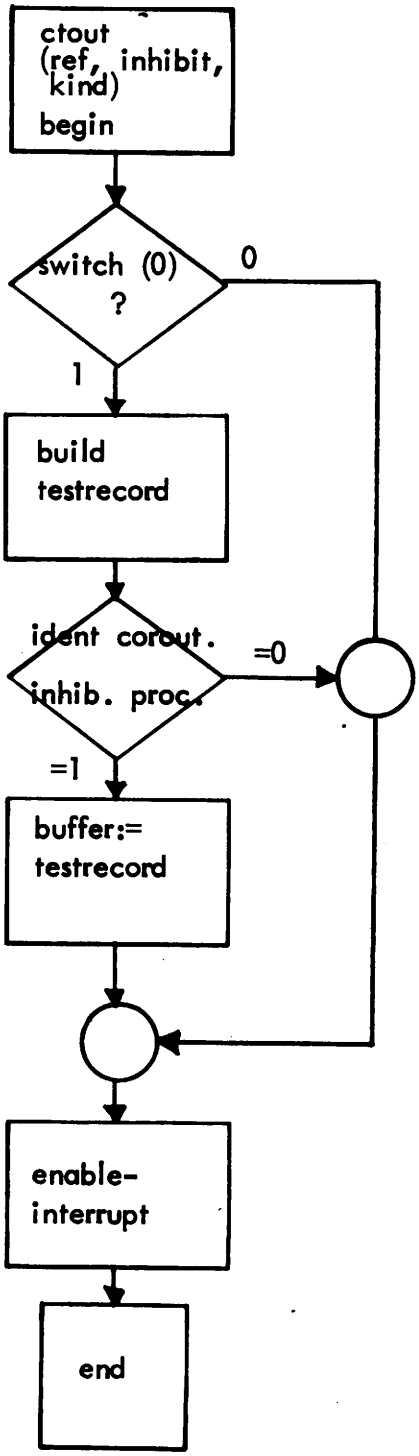
	call:	return:
ac0		undefined
ac1		undefined
ac2	addr. of ref.-1	undefined
ac3	link	addr. of coroutine-description

8.3. CTEST



	call:	return:
ac0		unchanged
ac1		unchanged
ac2	addr. of ref.-1	unchanged
ac3	link	addr. of coroutine-description

8.4. CTOUT



	call:	return:
ac0		unchanged
ac1		unchanged
ac2	addr. of ref.-1	unchanged
ac3	link	addr. of coroutine-description
link	long < 15 + inhibit < 8 + kind	