
RCSL No: 31-D697

Edition: May 1983

Author: Torben Steen Hansen

Title:

RC8000 MONITOR, PART 2
Basis System Version 2
Reference Manual

Keywords:

RC8000, Basic Software, Monitor, Monitor Procedures, Monitor Tables, Process Descriptions, Backing Storage Administration.

Abstract:

This manual describes monitor conventions, monitor procedures, format of monitor tables and process descriptions.

(134 printed pages)

FOREWORD

First edition: RCSL No 31-D697.

This is valid for all computers in the RC8000 family.

This manual is only valid for monitor release 9.0. A description of monitors older than release 9.0 is found in RCSL No 31-D679: RC8000 Monitor, Part 2, Basis System Version 1, Reference Manual.

A general description of the monitor is found in RCSL No 31-D476: RC8000 Monitor, Part 1, System Design.

The major changes in monitor release 9.0 concern:

- monitor procedures 28, 30, 32 and 34: they are now released for future use.
- as the maximum number of entries in a catalog has been increased, the format of the chaintables has been changed and the format of bs-claims of an internal process has been changed.
- the format of external processes has been changed because the maximum number of internal processes has been increased.

Torben Steen Hansen

A/S REGNECENTRALEN af 1979, April 1983

TABLE OF CONTENTS	PAGE
1. GENERAL MONITOR CONVENTIONS	1
1.1 Monitor Call	1
1.2 Interrupt Handling	2
1.3 Function Mask	3
1.4 Names	4
1.4.1 Names in Aux Catalog	6
1.4.2 Namekey	6
1.5 Protection and Visibility, Selection of Name	8
1.6 Permkey	10
1.6.1 Actual use of Permkey and Entry Base	11
1.7 Area Allocation, Slices	12
1.8 Strategy for Slice Allocation	13
1.9 State-Transitions and use of Document-State	14
2. DEFINITION OF MONITOR PROCEDURES	16
2.0 Procedure Set Interrupt, 0	17
2.1 Procedure Set Escape, 1	18
2.4 Procedure Process Description, 4	19
2.6 Procedure Initialize Process, 6	20
2.8 Procedure Reserve Process, 8	22
2.10 Procedure Release Process, 10	24
2.12 Procedure Include User, 12	25
2.14 Procedure Exclude User, 14	26
2.16 Procedure Send Message, 16	27
2.18 Procedure Wait Answer, 18	29
2.20 Procedure Wait Message, 20	31
2.22 Procedure Send Answer, 22	32
2.24 Procedure Wait Event, 24	33
2.26 Procedure Get Event, 26	35
2.28 Procedure Test Users and Reserver, 28	36
2.36 Procedure Get Clock, 36	37
2.38 Procedure Set Clock, 38	38
2.40 Procedure Create Entry, 40	39
2.42 Procedure Look Up Entry, 42	41
2.44 Procedure Change Entry, 44	42
2.46 Procedure Rename Entry, 46	44
2.48 Procedure Remove Entry, 48	45

TABLE OF CONTENTS (continued)	PAGE
2.50 Procedure Permanent Entry, 50	46
2.52 Procedure Create Area Process, 52	48
2.54 Procedure Create Peripheral Process, 54	49
2.56 Procedure Create Internal Process, 56	50
2.58 Procedure Start Internal Process, 58	52
2.60 Procedure Stop Internal Process, 60	53
2.62 Procedure Modify Internal Process, 62	55
2.64 Procedure Remove Process, 64	56
2.66 Procedure Test Event, 66	59
2.68 Procedure Generate Name, 68	61
2.70 Procedure Copy, 70	62
2.72 Procedure Set Catalog Base, 72	63
2.74 Procedure Set Entry Base, 74	64
2.76 Procedure Lookup Head and Tail, 76	65
2.78 Procedure Set Backing Storage Claims, 78	66
2.80 Procedure Create Pseudo Process, 80	68
2.82 Procedure Regret Message, 82	69
2.84 Procedure General Copy, 84	70
2.86 Procedure Look up Aux Entry, 86	72
2.88 Procedure Clear Statistics in Aux Entry, 88	73
2.90 Procedure Permanent Entry in Auxiliary Catalog, 90	74
2.92 Procedure Create Entry Lock Process, 92	75
2.94 Procedure Set Priority, 94	76
2.96 Procedure Relocate Process, 96	77
2.98 Procedure Change Address Base, 98	79
2.102 Procedure Prepare Backing Storage, 102	80
2.104 Procedure Insert Entry, 104	83
2.106 Procedure Insert Backing Storage, 106	86
2.108 Procedure Delete Backing Storage, 108	87
2.110 Procedure Delete Entries, 110	88
2.112 Procedure Connect Main Catalog, 112	89
2.114 Procedure Remove Main Catalog, 114	91
2.118 Procedure Lookup bs Claims, 118	92
2.120 Procedure Create Aux Entry and Area Process, 120 .	93
2.122 Procedure Aux Entry, 122	95
2.124 Procedure Send Pseudo Message, 124	96
2.126 Procedure Set cpa, 126	97

TABLE OF CONTENTS (continued)	PAGE
-------------------------------	------

3. IMPLEMENTATION DETAILS	98
3.1 Administration of Queues	98
3.2 Administration of Time Slices	99
3.3 Administration of Message Buffers	99
3.3.1 Message Buffer Pool	99
3.3.2 Message Buffer Format	100
3.3.3 Message Buffer States	101
3.4 Administration of Process Descriptions	102
3.5 Format of Internal Process Description	104
3.6 Format of External Process Description	109
3.6.1 Format of Peripheral Process Description ...	110
3.6.2 Format of Area Process Description	111
3.7 Format of Pseudo Process Description	112
3.8 Format of a Backing Storage Document	113
3.8.1 Format of Chaintable, as it is on a BS- Document	113
3.8.2 Format of a Chaintable, as it is in the Core-Store	115
3.9 Format of a Catalog	117
3.10 Format of a Catalog Entry	117
3.11 Format of Monitor Tables	119
3.12 Kinds of External Processes	121

APPENDIX:

A. REFERENCES	123
---------------------	-----

1. GENERAL MONITOR CONVENTIONS

1.

This chapter explains the general rules of monitor calls, internal interruption, privileged functions, names and catalog protection.

1.1 Monitor Call

1.1

The monitor is called within an internal process by the execution of an instruction with the format:

```
jd 1<11+<unsigned integer>
```

after loading parameter values in the working registers.

The monitor checks that the <unsigned integer> is the number of a monitor procedure.

Internal interrupts not recognized as monitor calls cause immediate switching to the interrupt procedure of the internal process (section 1.2).

The monitor also checks the validity of procedure parameters. The basic rule is that the addresses of stored parameters must be within the calling process; otherwise the monitor provokes an internal interruption of the process.

The following example shows the notation used to define the call of a monitor procedure:

```
lookup entry (name address, tail address, result)
w0  result (return)
w1  tail address (call)
w2
w3  name address (call)
jd 1<11+42
```

name address: entry name (call)
 tail address: number of segments (return)
 +2... 9 optional words (return)

The procedure lookup entry is called by executing the instruction `jd 1<11+42` with registers `w1` and `w3` containing the addresses of two storage areas, called tail and name, within the calling process. The name area contains an entry name when the procedure is called.

Upon return from the monitor procedure, register `w0` contains a result parameter. The monitor has stored a parameter, called number of segments and 9 optional words in the tail area.

Only the contents of registers and storage areas marked "(return)" have been changed by the procedure at the return from the monitor.

1.2 Interrupt Handling

1.2

An internal process can call the monitor and define the address of an interrupt procedure to be called when the process provokes an internal interrupt not recognized as a monitor call.

When the interrupt occurs, current register values are stored in the head of the interrupt procedure and execution of the internal process continues within the interrupt procedure:

interrupt address:	working register 0
+ 2	working register 1
+ 4	working register 2
+ 6	working register 3
+ 8	exception register
+10	instruction counter
+12	interrupt cause
+14	sb register
+16	(first instruction executed after interruption)

When the interrupt procedure is entered the contents of the four working registers are undefined.

The interrupt cause may have one of the following values:

- 0 protection violation
- 2 integer overflow
- 4 floating-point overflow/underflow
- 6 parameter error in monitor call
- 8 breakpoint forced by parent
- 10 zone stack error

(Further description is found in ref. [1], section 3.5).

An interrupt address equal to zero indicates that no interrupt procedure is defined by the process. If such a process provokes an internal interrupt, the process is removed from the timer queue and set in the state running after error. The process does not receive any more computing time in this state, but from the point of view of other processes it is still an existing process. The parent of the process can, however, reactivate it by means of stop and start.

An internal process is created with the interrupt address zero and with arithmetic interrupts masked off.

An internal process can, however, call the monitor and define the interrupt mask bits corresponding to integer and floating-point overflow. If these interrupts are masked off, they will not cause internal interruption of the process. Other kinds of internal interrupts cannot be masked off (for further details see ref. [2]).

1.3 Function Mask

1.3

During creation of an internal process the parent specifies a function mask in which each bit defines whether the child process is allowed to call one of the privileged monitor procedures. At

present the function mask bits have the following meaning
(1=function call allowed, 0=function call forbidden):

bit 0:	(1 shift 11)	(not used)
bit 1:	(1 shift 10)	auxiliary catalog handling
bit 2:	(1 shift 9)	main catalog handling
bit 3:	(1 shift 8)	(not used)
bit 4:	(1 shift 7)	CREATE PERIPHERAL PROCESS
bit 5:	(1 shift 6)	REMOVE PERIPHERAL PROCESS
bit 6:	(1 shift 5)	auxiliary entry handling
bit 7:	(1 shift 4)	SET CLOCK
bit 8:	(1 shift 3)	(not used)
bit 9:	(1 shift 2)	(not used)
bit 10:	(1 shift 1)	(not used)
bit 11:	(1 shift 0)	(not used)

auxiliary catalog handling is: PREPARE BACKING STORAGE

INSERT ENTRY
INSERT BACKING STORAGE
DELETE BACKING STORAGE
DELETE ENTRIES
CONNECT MAIN CATALOG
REMOVE MAIN CATALOG

main catalog handling is: CONNECT MAIN CATALOG
REMOVE MAIN CATALOG

auxiliary entry handling is: CREATE AUX ENTRY AND AREAPROCESS
REMOVE AUX ENTRY

1.4 Names

1.4

A NAME is a textstring of 12 ISO characters (equal to 4 words) beginning with a small letter followed by at most 10 small letters or digits, terminated by NULL-characters.

Associated with a name is a BASE, consisting of two integers:
lowerbase and upperbase.

The base must conform to the rule:

$$-8388607 \leq \text{lowerbase} \leq \text{upperbase} \leq 8388606.$$

Names (with associated bases) are used to identify processes and catalog entries.

Two names (with associated bases) are IDENTICAL if:

NAME 1 = NAME 2
 and LOWERBASE 1 = LOWERBASE 2
 and UPPERBASE 1 = UPPERBASE 2

Two names (with associated bases) are OVERLAPPING if:

NAME 1 = NAME 2
 and
 lowerbase 1 < lowerbase 2 $\leq$ upperbase 1 < upperbase 2
 or
 lowerbase 2 < lowerbase 1 $\leq$ upperbase 2 < upperbase 1

A name (with associated base) is UNIQUE among a specified set of names, if it is not identical to, or overlapping any of the names from the specified set.

A PROCESS NAME (with associated base) must be unique among all processes (and all entries in the main catalog, if the maincatalog is present as the name and base of an entry are used to name the external process describing the entry).

A MAINCATALOG ENTRY NAME (with associated base) must be unique among all processes (and all entries in the main catalog, if the main catalog is present).

An AUXCAT ENTRY NAME (with associated base) must be unique among all entries in that auxiliary catalog.

A GENERATED NAME has the format:

wrk<6 digits>

followed by three NULL-characters.

A generated name is completely unique (i.e. independent of base) among all processes (and all entries in the main catalog, if the main catalog is present and therefore in all connected aux catalogs).

1.4.1 Names in Aux Catalog

1.4.1

When a disc is mounted containing an aux catalog, the entries in this catalog are inserted (by means of INSERT ENTRY) in the main catalog.

If a name (with associated base) is already in the main catalog, the entry from the aux catalog cannot be inserted in the main catalog.

This may cause the following inconvenience. If the entry in the main catalog is removed and another entry with the same name (with associated base) is inserted (by means of PERMANENT ENTRY, RENAME ENTRY, SET ENTRY BASE) in the aux catalog, the old entry will be removed.

1.4.2 Namekey

1.4.2

Namekey is a hashed value of the entry name and may be computed as follows:

begin

```
long sum, part_1_of_name, part_2_of_name;
integer namekey, catalogsize, noofkeys, segmentno;
```

```

part_1_of_name := bit (0:47) of entry name
part_2_of_name := bit (48:95) of entry name
catalogsize    := size of catalog
noofkeys       := no of keys per partition;

sum            := part_1_of_name + part_2_of_name;
sum            := sum shift (-24) + sum extract 24;
sum            := sum extract 24 + (sum shift (-12) shift 36)/(2**36);
sum            := sum extract 24;
segmentno     := sum mod catalogsize;
namekey       := segmentno mod noofkeys;
end;

```

In SLANG the algorithm looks like:

```

dl  w2  part_1_of_name
aa  w2  part_2_of_name
wa  w2  2
ba  w2  4
al  w1  0
wd  w2  catalogsize
; w1 = segmentno
al  w3  x1
al  w2  0
wd  w3  noofkeys
; w2 = namekey

```

The namekey and segment no are used when a catalog is searched for an entry with a specific name.

In a catalog with 12 segments and number of keys = 6 (the catalog is divided into two partitions) there will be the following connection between segment number and namekey:

```

segmentno: 0  1  2  3  4  5  6  7  8  9 10 11
namekey:   0  1  2  3  4  5  0  1  2  3  4  5

```

The search for an entry starts at the segment given by segmentno and continues cyclically through the catalog (last segment is followed by segment 0) until segment = segmentno + noofkeys - 1.

Only if noofkeys is equal to catalogsize overflow is allowed through the whole catalog.

If the catalog contains more than one partition overflow is only allowed within noofkeys segments. Therefore it may occur that an entry cannot be inserted in the catalog even if there are unused entries in the catalog.

1.5 Protection and Visibility, Selection of Name

1.5

Associated with each internal process are two PROTECTION BASES, namely the MAXBASE and the STDBASE and a VIEWPOINT BASE, namely the CATBASE.

(These bases are initialized by the parent, when the internal process is created).

The catbase may freely be changed by the internal process, as long as the following is fulfilled:

lower stdbase $\leq$ lower catbase $\leq$ upper catbase $\leq$ upper stdbase
or

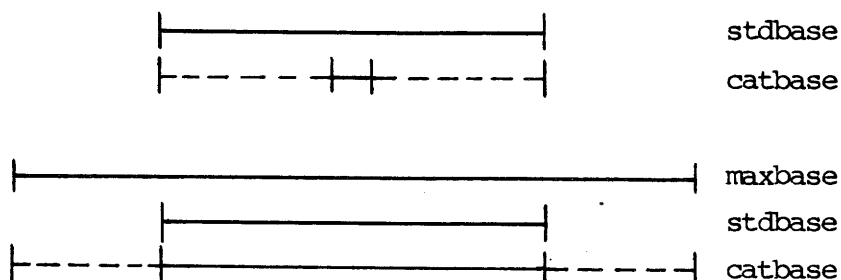
lower maxbase $\leq$ lower catbase $\leq$ lower stdbase

and

upper stdbase $\leq$ upper catbase $\leq$ upper maxbase,

i.e. "inside" stdbase or "between" stdbase and maxbase.

change of catalogbase:



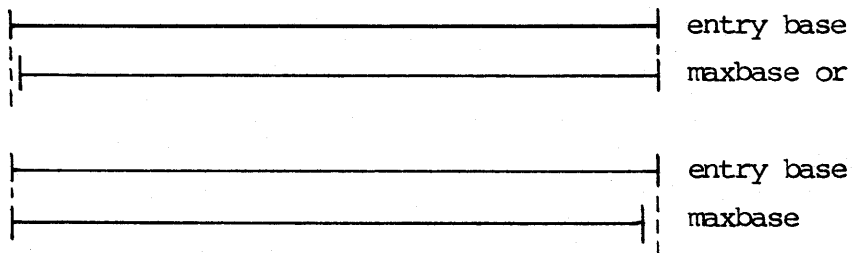
A maincatalog entry (and the corresponding auxiliary catalog entry) is PROTECTED against an internal process if:

lower base < lower maxbase $\leq$ upper maxbase $\leq$ upper base
or

lower base $\leq$ lower maxbase $\leq$ upper maxbase < upper base

i.e. maxbase is "inside" entry base.

Protected:

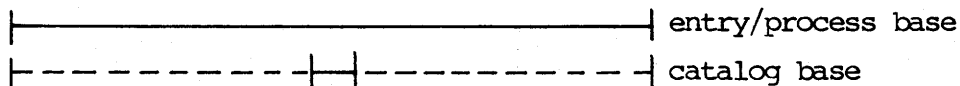


A process (or catalog entry) is VISIBLE to an internal process if:

lower base $\leq$ lower catbase $\leq$ upper catbase $\leq$ upper base,

i.e. catbase "inside" base of process (or catalog entry).

Visible:



An internal process may only read data areas, whose catalog entries are visible to the internal process.

An internal process may only modify (or remove) those catalog entries (and their associated data areas) which are visible to - but not protected against - the internal process.

A process (or catalog entry), with name NAME1, is BETTER than

another process (or catalog entry), with name NAME2, when:

both processes (or catalog entries) are visible
 and
 $NAME1 = NAME2$
 and
 $lowerbase2 < lowerbase1$
 or
 $upperbase1 < upperbase2$.

Monitor procedures like INITIALIZE/RESERVE/RELEASE PROCESS, SEND MESSAGE, REMOVE PROCESS etc. refer to the best process with the given name.

Monitor procedures like CHANGE/RENAME/REMOVE ENTRY etc. refer to the best catalog entry with the given name.

1.6 Permkey

1.6

Associated with each catalog entry is a PERMKEY.

The permkey is an integer obeying:

$$0 \leq \text{permkey} \leq \text{max permkey}.$$

The permkey may be used by an operating system to determine the "degree of survival" for a catalog entry.

When a catalog entry is created, its permkey is initialized to zero, indicating a temporary entry.

The permkey of an entry may be changed by an internal process using the procedures:

PERMANENT ENTRY

or PERMANENT ENTRY IN AUXILIARY CATALOG

As long as a catalog entry has a permkey,

$$0 \leq \text{permkey} < \text{min_auxcat_permkey}$$

the catalog entry is only present in the main catalog.

The catalog entry will be present in both maincatalog and auxiliary catalog, when:

$$\text{min_auxcat_permkey} \leq \text{permkey} \leq \text{max_permkey}$$

It means that - from the monitor's point of view - a catalog entry is considered as a temporary entry, when its permkey is less than min_auxcat_permkey.

1.6.1 Actual use of Permkey and Entry Base

1.6.1

```
min_auxcat_permkey = 2
max_permkey        = 3
min_global_permkey = 3
```

Permkey together with the entry base represents the so-called scope of an entry. The operating system BOSS has introduced special scopes (see ref. [5]).

<u>scope</u>	<u>permkey</u>	<u>entrybase</u>
temp	0	stdbase
login	2	stdbase
project	3	maxbase
user	3	def. by operating system (usually between std and max)

When the operating system s creates a child process, catbase is set to stdbase.

When the operating system boss creates a job, catbase is set surrounding stdbase and included in maxbase.

When fp initializes, userbase is set to catbase and catbase is set to stdbase.

e.g. max = 0-100; std = 11-11

<u>s</u>	<u>boss</u>
catbase := 11-11	catbase := 10-20 (from the boss catalog)

<u>fp</u>	<u>fp</u>
userbase := 11-11	userbase := 10-20
catbase := 11-11	catbase := 11-11

1.7 Area Allocation, Slices

1.7

Each document is divided into a number of so-called slices each consisting of an integral number of segments. To each slice corresponds a halfword in core store in the so-called chaintable. A backing storage area consists of an integral number of slices and the corresponding halfwords of core store are linked together. In this way the segments in an area need not be physically consecutive and therefore it is always possible to allocate an area if the necessary number of slices are free. It is also possible to extend an area by a call of change entry. As the chaintable is essential for interpretation of the contents of a drum or disc, a copy of the chaintable is kept on the corresponding device. This copy is updated every time a catalog entry tagged with a permkey larger than or equal to min_auxcat_permkey is changed. This and the corresponding rule concerning auxiliary catalog means that only the 'most permanent' files will survive a system breakdown or removal of a disc pack.

The amount of backing storage that an internal process uses is a resource that should be within the control of the operating system. This is achieved in a similar way as the control of the usage of message buffers, area processes etc.: by associating with each internal process the maximum amount of backing storage it may use. These claims specify for each backing storage document and each permkey the number of catalog entries and the number of slices the process may tag with this key or a larger key.

Use of entries are registered as use of main catalog entries for keys less than `min_auxcat_permkey` and as use of entries in both main catalog and in the auxiliary catalog for keys larger than or equal to `min_auxcat_permkey`.

The backing storage claims of an internal process may be changed (increased or decreased) by the parent by means of a monitor procedure set backing storage claims.

1.8 Strategy for Slice Allocation

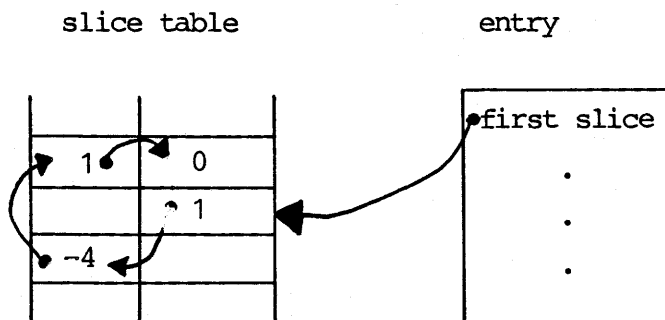
1.8

The algorithm for allocating slices to an area (either when creating a new area or extending an existing area) will always try to keep the area as a contiguous whole.

New areas are preferably allocated as consecutive slices, if not possible unused slices from the beginning of the chaintable will be used.

The strategy for extension of areas is as follows. First, unused slices in extension of the logical last slice will be used. Secondly, the unused slices within the physical first and last slice will be used. And finally, unused slices from the beginning of the chaintable will be used.

Slice allocation:



A free slice is marked with the value -2048 in the corresponding chaintable element in core store.

A link in a slice chain in the chaintable is the (relative) ad-

dress of the next slice-element in the chain.

1.9

The link is zero in the last slice chain element.

See also subsections 3.8.1 and 3.8.2.

1.9 State-Transitions and use of Document-State

Each chaintable has a document-state which determines the "legality" of several monitor calls.

The possible states are:

- a. idle : 1
- b. after prepare : 2
- c. during insert : 4
- d. ready : 8
- e. during delete : 16
- f. during aux : 64

The following table lists these monitor-calls and the state(s) they demand for the document they involve:

monitor call	document state
create entry	: ready (+ maincat present)
lookup entry	: (indifferent) (+ maincat present)
change entry	: ready (+ maincat present)
rename entry	: ready (+ maincat present)
remove entry	: ready (+ maincat present)
permanent entry	: ready (+ maincat present)
create area process	: after prepare during insert ready during aux
set entry base	: ready (+ maincat present)
lookup entry head and tail	: (indifferent) (+ maincat present)
set bs claims	: after prepare during insert ready during delete during aux

<u>monitor call</u>	<u>document state</u>
permanent entry in	
aux catalog	: ready (+ maincat present)
create entry lock	
process	: after prepare
	during insert
	ready
	during aux
prepare bs	: idle
insert entry	: after prepare
	during insert
insert bs	: after prepare
	during insert
delete bs	: after prepare
	during insert
	ready
	during delete
	during aux
delete entries	: during delete
connect main catalog:	after prepare (+ maincat not present)
	during insert
remove main catalog	: (indifferent) (+ maincat present)
create aux entry	: after prepare
	during insert
	during aux
remove aux entry	: after prepare
	during insert
	during aux

The following monitor-calls may change the document state:

<u>monitor call</u>	<u>new document state</u>
prepare bs	: after prepare
insert entry	: during insert
insert bs	: ready
delete bs	: during delete
delete entries	: idle (when all maincat has been searched)
connect main catalog:	during insert
create aux entry	: during aux
remove aux entry	: during aux

2. DEFINITION OF MONITOR PROCEDURES

2.

The following defines the monitor procedures in the monitor by giving for each procedure the parameters, the description and the error reactions. In describing the error reactions the following terminology is used:

Document not ready: The state of the document named in the document name of an area entry is not 'ready' (see section 1.9).

Name conflict: It is attempted to introduce a name twice in the catalog with base intervals (a,b) and (a', b') so that either $a=a'$ and $b=b'$ or $a < a' \leq b < b'$. (See section 1.4).

2.0 Procedure Set Interrupt, 0

2.0

```
set interrupt (interrupt_address, interrupt_mask)
w0 interrupt mask (call)
w1
w2
w3 interrupt address (call)
jd 1<11 + 0
```

interrupt address: (see section 1.2)

Defines the internal interrupt address of the calling process and the interrupt mask bits 1 and 2 for integer and floating-point overflow, the remaining mask bits are irrelevant in the call; they are always set to a standard value by the monitor.

The interrupt address must either be zero or point to an area within the calling process.

PARAMETER ERROR: interrupt area outside calling process.

2.1 Procedure Set Escape, 1

2.1

Set escape (escape-address, escape-specification)

w0 escape-specification (call)

w1

w2

w3 escape-address (call)

jd 1<11+1

Defines the escape address, escape mode and escape mask of the calling process.

Sets the escape-specification = escape-mode and escape-mask in the exception register (status register).

Escape-specification:

escape-mode <22+ escape-mask <12

escape-mode 1 bit (no. 1)	0 no escape function
	1 escape function

escape-mask 6 bits (no. 6-11)

The escape address must either be zero or point to an area within the calling process. If zero no escape function is performed.

Setting escape-specification or escape address to zero the escape function is inactive.

PARAMETER ERROR: escape address outside calling process.

```
process description (name address, result)
w0 result (return)
w1
w2
w3 name address (call)
jd 1<11+4
```

name address: process name (call)

Checks the existence of a process with a given name. The result is the address of the process description within the monitor, if the process exists; otherwise the result is zero.

```
result =0 process does not exist
       >0 process description address
```

PARAMETER ERROR: process name area outside calling process.

2.6 Procedure Initialize Process, 6

2.6

```

initialize process (name address, result)
w0 result (return)
w1
w2
w3 name address (call)
jd 1<11+6

```

name address: process name (call)

Checks the existence of a process with a given name and prepares it for communication with the calling process. Messages received by the process before initialization will still be processed and answered.

The initializing depends on the kind of the process as follows:

INTERNAL PROCESS, CLOCK, TYPEWRITER or PSEUDO PROCESS:

Only the existence of the process is checked.

PAPER TAPE READER, PAPER TAPE PUNCH: The existence of the peripheral process with the calling process as a user is checked. The device is reserved unless it is reserved by another process.

The procedure prepares the peripheral process so that input/output in flexo-mode is assumed to be in lower case.

OTHER PERIPHERAL PROCESS: The existence of the peripheral process with the calling process as a user is checked. If the peripheral process controls a sequential document, (e.g. card reader, magnetic tape) it is reserved for the calling process unless another process has reserved it. Further details, see ref. [4].

If the external process represents a temporary link the calling process is included as user of the process.

AREA PROCESS: The existence of the area process is checked. If the calling process is not a user of the area process, result 3 is returned. After initialization, only input from the area is allowed, unless the process has already been reserved by the internal process.

result = 0 process initialized

1 reserved by another process

2 calling process is not a user

3 process does not exist;

calling process is not a user of area process.

PARAMETER ERROR: process name area outside calling process.

```

reserve process (name address, result)
w0 result (return)
w1
w2
w3 name address (call)
jd 1<11+8

```

```

name address: process name (call)

```

Reserves a process with a given name for exclusive communication with the calling process.

Messages received by the process before reservation will still be processed and answered.

Reservation depends on the kind of the process as follows:

INTERNAL PROCESS, CLOCK or PSEUDO PROCESS: Reservation is not allowed.

TYPEWRITER: The existence of the peripheral process with the calling process as a user is checked. It is reserved for the calling process unless another process has reserved it. When the typewriter is reserved, the 'att' request will be suppressed at succeeding pushes of the attention key and the attention-message will be sent directly to the reserver.

PAPER TAPE READER, PAPER TAPE PUNCH: The existence of the peripheral process with the calling process as a user is checked. The device is reserved unless it is reserved by another process.

The procedure prepares the peripheral process so that input/output in flexo-mode is assumed to be in lower case.

OTHER PERIPHERAL PROCESS: The existence of the peripheral process with the calling process as a user is checked. It is reserved

for the calling process unless another process has reserved it. Further details, see ref. [4].

AREA PROCESS: The existence of the area process is checked. If the calling process is not a user of the area process, result 3 is returned. The area process is reserved for the calling process provided no other process has reserved it, and provided the area is not protected against the calling process, see section 1.5. After reservation both input and output to the area are allowed.

result = 0 process reserved

1 reserved by another process

2 calling process is not a user;
process cannot be reserved

3 process does not exist;
calling process is not a user of area process.

PARAMETER ERROR: process name area outside calling process.

2.10 Procedure Release Process, 10

2.10

release process (name address)

w0

w1

w2

w3 name address (call)

jd 1<11+10

name address: process name (call)

Releases a possible reservation of a process with a given name.

Release depends on the kind of the process as follows:

INTERNAL PROCESS, PSEUDO PROCESS: Nothing is done.

EXTERNAL PROCESS: If the external process exists and is reserved by the calling process, the reservation is cancelled.

Messages received by the process before release will still be processed and answered.

PARAMETER ERROR: process name area outside calling process.

2.12 Procedure Include User, 12

2.12

```
include user (name address, device number, result)
w0 result (return)
w1 device number (call)
w2
w3 name address (call)
jd 1<11+12
```

name address: process name (call)

Includes an internal process with a given name as a user of a peripheral device with a given number. The internal process must be a child of the calling process and the latter must be a user of the device.

```
result = 0 child included as a user
          2 calling process is not a user
          3 process does not exist;
            process is not an internal process;
            process is not a child of calling process
          4 device number does not exist
```

PARAMETER ERROR: process name area outside calling process.

2.14 Procedure Exclude User, 14

2.14

```

exclude user (name address, device number, result)
w0 result (return)
w1 device number (call)
w2
w3 name address (call)
jd 1<11+14

```

name address: process name (call)

Excludes an internal process with a given name as a user and reserver of a peripheral device with a given number. The internal process must be a child of the calling process and the latter must be a user of the device.

Messages received by the peripheral process before exclusion of the user will still be processed and answered.

```

result = 0 child excluded as a user
        2 calling process is not a user
        3 process does not exist;
          process is not an internal process;
          process is not a child of calling process
        4 device number does not exist

```

PARAMETER ERROR: process name area outside calling process.

send message (name address, message identification, message
address, buffer address)

w0

w1 message address (call)

w2 message identification (call), buffer address (return)

w3 name address (call)

jd 1<11+16

name address: process name (call)

+8 name table address (return)

message address: 8 words (call)

Selects an available message buffer, decreases the buffer claim of the calling process by one and copies a message of eight words into the buffer. The message is delivered in the queue of a receiving process with a given name. The receiving process is activated if it is waiting for a message. If the receiving process is waiting for an event it is activated if it has buffer claims enough (see wait event). The calling process continues after being informed of the address of the message buffer.

The message identification is saved in the message buffer (in buffer address -2), and is available after a call of test event.

If the receiving process does not exist (maybe because of illegal name format), the monitor generates a dummy answer to the message as described under wait answer, section 2.18.

If the receiving process represents a temporary link (kind=85) the calling process is inserted as user of the process.

The format and interpretation of a message depend on the kind of the receiving process. For external process details are given in ref. [4].

After a call of send message the monitor stores an address, the name table address, after the process name. This address is used

in subsequent calls of send message to speed up identification of the receiver. The name table address can be destroyed by the calling process, but this will slow down the next call of send message.

Note that the name table address has priority over the catalog hierarchy in determining to which process the message is sent. This means that in certain cases the destruction of the name table address will not only slow down the call of send message but may also lead to another result (because the hierarchy may have changed).

The process description address of the receiver can be loaded by indirect addressing of the name table address, for example:

```
r1 w1 (name address+8); w1 := process descr addr
```

If the claim of the calling process is exceeded the buffer address is zero:

```
buffer address =0 buffer claim exceeded
                >0 selected buffer address
```

PARAMETER ERROR: process name area or message area outside calling process.

```

wait answer (buffer address, answer address, result)
w0 result (return)
w1 answer address (call)
w2 buffer address (call)
w3
    jd 1<11+18

```

answer address: 8 words (return)

Delays the calling process until an answer arrives in the given message buffer. On arrival the answer, consisting of 8 words, is copied into the calling process. The message buffer is released and the buffer claim of the calling process is increased by one.

The message buffer must be one that has been assigned to the calling process during a previous call of send message.

The format of an answer depends on the kind of the process that has received and answered the message. For external processes details are given in ref. [4].

The result specifies whether a normal or dummy answer was received. A normal answer is delivered by a process in response to a received intelligible message. A dummy answer is generated by the monitor when a message is addressed to a non-existent process. A dummy answer can also be generated by the receiving process in response to an undesired or unintelligible message or when a malfunction (such as a disconnected peripheral device) prevents completion of an operation specified in a message.

```

result = 1 normal answer
        2 dummy answer, message rejected
        3 dummy answer, message unintelligible
        4 dummy answer, receiver malfunction
        5 dummy answer, receiver does not exist

```

PARAMETER ERROR: buffer address does not point at a message buffer assigned to calling process; answer area outside calling process.

```
wait message (name address, message address, buffer address,
              result)
```

```
w0 result (return)
```

```
w1 message address (call)
```

```
w2 buffer address (return)
```

```
w3 name address (call)
```

```
jd 1<11+20
```

```
name address: process name (return)
```

```
message address: 8 words (return)
```

Delays the calling process until a message arrives in its queue. On arrival the name of the sending process and the message, consisting of 8 words, are copied into the calling process. The address of the message buffer in which the message was transmitted is delivered as a return parameter. The buffer is removed from the queue and is now ready to transmit an answer.

The calling process must supply a message buffer (i.e. the buffer claim of the calling process is decreased by one) if it has not already done so (if the buffer address is equal to the buffer address returned by wait event, the calling process supplied the buffer when it called wait event).

The result is the process description address of the sender. If the message has been regretted, the result is the negative process description address of the sender. In this case a dummy sender name is copied into the calling process.

```
result =0 buffer claim exceeded
```

```
>0 process description address
```

```
<0 process description address, if message regretted
```

```
buffer address =0 buffer claim exceeded
```

```
>0 selected buffer address
```

PARAMETER ERROR: name area or message area outside calling process.

```

send answer (result, answer address, buffer address)
w0 result (call)
w1 answer address (call)
w2 buffer address (call)
w3
jd 1<11+22

```

answer addresss: 8 words (call)

Copies an answer of eight words into a message buffer in which a message has been received and increases the buffer claim of the calling process by one.

If the message buffer is regretted by the original sender (or the sender has been removed) the message buffer is merely released.

Otherwise the message is delivered in the queue of the original sender. The sender of the message is activated if it is waiting for an answer or event. The result defined by the calling process is delivered to the sender as the result of wait answer; it specifies whether the answer is normal or dummy.

```

result = 1 normal answer
          2 dummy answer, message rejected
          3 dummy answer, message unintelligible
          4 dummy answer, receiver malfunction
          5 dummy answer, receiver does not exist

```

PARAMETER ERROR: buffer address does not point at a message buffer received by calling process; answer area outside calling process; illegal result value.

(Note: result 5 = receiver does not exist is legal for simulation purposes).

```

wait event (previous buffer address, next buffer address,
            result)
w0 result (return)
w1
w2 previous buffer address (call)
   next buffer address (return)
w3
   jd 1<11+24

```

Delays the calling process until either a message (for which the process has claims) or an answer arrives in its queue after the buffer given by the previous buffer address. The process is supplied with the address of the next buffer and a result indicating whether it contains a message or an answer. If the previous buffer address is zero, the queue is examined from the start.

The wait event procedure does not remove the next buffer from the queue. This must be done by one of the procedures wait answer, wait message or get event.

If the queue of the calling process contains a message for which the calling process has already supplied a buffer (in a previous call of wait event) the buffer supplied by the calling process is released and the buffer claim of the calling process is increased by one. The calling process must supply a message buffer (i.e. the buffer claim of calling process will be decreased by one) if next buffer address points to a message.

If the buffer claim of the process is zero it will only be started if an answer arrives.

```

result = 0 message, claimed
        1 answer
        2 interrupt
        4 immediate operation
(result 2 and 4 is only possible for a driver process).

```

PARAMETER ERROR: previous buffer address does not point at a message buffer in the queue of the calling process.

```

get event (buffer address)
w0
w1
w2 buffer address (call)
w3
jd 1<11+26

```

Removes a given buffer from the queue of the calling process. If the buffer contains a message, it is made ready for the sending of an answer. In this case the calling process must supply a message buffer (i.e. the buffer claim of the calling process is decreased by one) if it has not already done so (see wait message section 2.10). If the buffer contains an answer, it is released and the buffer claim of the calling process is increased by one.

The copying of information from the buffer (sender description address, message or answer and result) must be done by the calling process itself before get event is called. This requires a knowledge of the format of a message buffer (see subsection 3.3.2).

PARAMETER ERROR: buffer address does not point at a message buffer in the queue of the calling process.

Test users and reserver (internal name address, external name address, answer, result).

w0 result (return)
 w1 internal name address (call)
 w2 answer (return)
 w3 external name address (call)

jd 1<11+28

Internal name address: name of internal process (call)

External name address: name of external process (call)

Test whether an internal process is user or reserver of an external process, and if there are other users or another reserver.

If the first word of the name of the internal process is zero, the call refers to the calling process.

The success of the call is signalled in result where

result	
0	ok, further information in w2
3	internal process does not exist
4	external process does not exist

The answer is signalled in w2 in bit 20-23, where

bit 23 = 1 means internal process is user
 bit 22 = 1 means internal process is reserver
 bit 21 = 1 means other user(s)
 bit 20 = 1 means another process is reserver.

2.36 Procedure Get Clock, 36

2.36

```
get clock (clock)
w0 clock(0:23)(return)
w1 clock(24:47)(return)
w2
w3
jd 1<11+36
```

Delivers an updated value of the 48-bit clock with unit position
0.1 millisecond, counted from 00.00.00, 01. jan. 1968

2.38 Procedure Set Clock, 38

2.38

```
set clock (clock)
w0 clock(0:23)(call)
w1 clock(24:47)(call)
w2
w3
jd 1<11+38
```

Initializes the 48-bit clock with unit position 0.1 millisecond.
This is a privileged function.

It should be noted that clock values stored in the process descriptions of internal processes (start time and wait time) are unchanged after a call of set clock (see section 3.5).

PARAMETER ERROR: function forbidden in calling process.

```

create entry (name address, tail address, result)
w0 result (return)
w1 tail address (call)
w2
w3 name address (call)
jd 1<11+40

```

```

name address: entry name (call and return)
tail address: number of segments (call)
+2          9 optional words (call and return)

```

Creates a new entry in the main catalog with name and tail as specified in the call. If the first word of the name is zero, the monitor generates a unique entry name and stores it in the name area specified in the call. The entry is created with the permkey zero and with an entry base equal to catalog base of calling process.

The catalog base of calling process may not be outside stdbase of calling process.

The bs-claims of the calling process are decreased as if the size is a multiple of the slice length.

If the number of segments is greater than or equal to 0, the entry is an area entry.

In this case the tail area is composed like this:

```

tail address: number of segments (call)
+2:
+4:          document name (call and return)
+6:
+8:
+10:         5 optional words (call)

```

The first word of document name may be 0 or 1 or the whole document name must be equal to a document name corresponding to a logic drum or a logic disc.

In the first case, the area is allocated on a document on which the calling process has sufficient claims (preferring a drum if word 2 is 0 and a disc if word 2 is 1). After the creation, the tail area is changed to contain the document name of the document on which it was allocated.

In the second case, the area is allocated on the document with the given name.

```

result = 0 entry created
      1 no room in catalog (see section 1.4.2)
      2 catalog i/o error;
        document not mounted or document not ready
      3 name conflict
      4 claims exceeded
      5 catalog base of calling process does not allow
        creation of entry, i.e. catalog base of calling
        process outside stdbase of calling process
      6 name format of entry or document illegal
      7 maincat not present

```

PARAMETER ERROR: name area or tail area outside calling process

```

lookup entry (name address, tail address, result)
w0 result (return)
w1 tail address (call)
w2
w3 name address (call)
jd 1<11+42

```

```

name address: entry name (call)
tail address: number of segments (return)
+2          9 optional words (return)

```

Looks up an entry in the main catalog with a given name and copies the entry tail into the calling process.

```

result = 0 entry looked up
        2 catalog input/output error
        3 entry does not exist
        6 name format illegal
        7 maincat not present

```

PARAMETER ERROR: name area or tail area outside calling process.

cf. lookup head and tail, jd 1<11+76

```
change entry (name address, tail address, result)
```

```
w0 result (return)
```

```
w1 tail address (call)
```

```
w2
```

```
w3 name address (call)
```

```
jd 1<11+44
```

```
name address: entry name (call)
```

```
tail address: number of segments (call)
```

```
+2          9 optional words (call)
```

Changes the tail of an entry in the main catalog with a given name provided the entry is not protected against calling process, see section 1.5.

A possible corresponding auxcat entry is changed in the auxiliary catalog.

If the entry describes an area on the backing store, the size of the area, given by the first word of the tail, can be reduced or increased. Reduction is from the upper end, that is, the first segments remain unchanged. The document name in the tail is not changed and has no influence. The bs-claims of the calling process is adjusted, as if the new and old area-size were multiples of the slice-length on the document.

If the entry does not describe an area, the first word of the new tail must be less than zero.

If the entry is used as a process, it must not be reserved by another internal process. The area process description is changed in accordance with the new size.

```
result = 0 entry changed
```

```
2 catalog i/o error;
```

```
document not ready
```

```
3 entry not found
```

- 4 entry protected, i.e. base of entry name not
contained in max base of calling process
- 5 area process (or entry lock process) reserved
by another process
- 6 name format or new size illegal; claims exceeded
- 7 maincat not present

PARAMETER ERROR: name area or tail area outside calling process.

```

rename entry (name address, new name address, result)
w0 result (return)
w1 new name address (call)
w2
w3 name address (call)
jd 1<11+46

```

```

name address: old entry name (call)
new name address: new entry name (call)

```

Changes the name of an entry in the main catalog provided the entry is not protected against the calling process, see section 1.5.

A possible corresponding auxcat entry is renamed in the auxiliary catalog.

This procedure leaves the catalog entry completely unchanged, except for the alteration of the entry name.

If the entry is used as a process, the calling process must be the only user. The name of the process is changed to the new name.

```

result = 0 entry renamed
      1 no room in catalog (see section 1.4.2)
      2 catalog i/o error;
        document not ready
      3 entry not found or new name overlap
      4 entry protected, i.e. base of entry name not
        contained in max base of calling process.
      5 area process (or entry lock process) used by
        another process
      6 name format (old or new name) illegal
      7 maincat not present

```

PARAMETER ERROR: old name area or new name area outside calling process.

```

remove entry (name address, result)
w0 result (return)
w1
w2
w3 name address (call)
jd 1<11+48

```

name address: entry name (call)

Removes an entry in the main catalog with a given name and its associated area on the backing store provided the entry is not protected against the calling process, see section 1.5.

A possible corresponding auxcat entry is removed from the auxiliary catalog.

The bs-claims of the calling process are increased, as if the entry-size was a multiple of the slicelength on the document.

If the entry is used as a process, it must not be used by another internal process. The area process description is removed and the area claim of the calling process is increased by one.

```

result = 0 entry removed
      2 catalog i/o error;
        document not ready
      3 entry not found
      4 entry protected, i.e. base of entry name not
        contained in max base of calling process
      5 area process (or entry lock process) used by
        another process
      6 name format illegal
      7 maincat not present

```

PARAMETER ERROR: name area outside calling process.

```

permanent entry (name address, permkey, result)
w0 result (return)
w1 perm key (call)
w2
w3 name address (call)
jd 1<11+50

```

```

name address: entry name (call)

```

Changes the perm key of an entry in the main catalog provided the entry is not protected against the calling process, see section 1.5.

The catalog entry is completely unchanged, except for the alteration of the perm key.

The bs-claims of the calling process is adjusted, as if the entry size was a multiple of slicelength.

```

new permkey>old permkey:

```

```

claims are decreased between old permkey + 1
and new permkey

```

```

old permkey>new permkey:

```

```

claims are increased between new permkey + 1
and old permkey.

```

If the entry is used as a process, it must not be reserved by another internal process.

If the old permkey is less than min_auxcat_permkey and the entry is not an area (i.e. the entry-size is less than 0), the associated auxiliary catalog is defined as the auxiliary catalog for the document on which the main catalog resides.

If the new permkey is less than min_aux_cat_perm_key, a possible auxcat entry is removed from the auxiliary catalog.

If the new permkey is greater than or equal to min_aux_cat_perm_key, a possible new auxcat entry is created in the auxiliary catalog.

If the new permkey is less than min_global_permkey, the base of the entry name must be contained in the standard base of the calling process, otherwise error 4 (=key illegal) is returned. Result = 3, name overlap is possible because the auxiliary catalog may contain an entry which has not been successfully inserted into the main catalog.

Cf. the procedure permanent entry in auxiliary catalog, jd
1<11+90.

result = 0 entry permanent

- 1 no room in catalog (see subsection 1.4.2)
- 2 catalog i/o error;
document not ready
- 3 entry not found; name overlap (in auxiliary
catalog)
- 4 entry protected, i.e. base of entry name not
contained in max base of calling process;
key illegal
- 5 entry reserved by another process
- 6 name format illegal; claims exceeded
- 7 maincat not present

PARAMETER ERROR: name area outside calling process.

```

Create area process (name address, result)
w0 result (return)
w1
w2
w3 name address (call)
jd 1<11+52

```

```

name address: process name (call)

```

Makes an area on the backing store available to the calling process as an external process with the same name and base as the corresponding catalog entry. The entry is looked up in the main catalog and described as a process within the monitor with the calling process as a user whose area claim is decreased by one. If the area process already exists, the calling process is defined as a user and its area claim decreased by one. (If the main catalog does not exist, the process name may be the name of an auxiliary catalog).

Cf.: create entry lock process 92.

```

result = 0 area process created
      1 area claims exceeded
      2 catalog i/o error;
        state of document does not permit this call
      3 entry not found
      4 entry does not describe an area
      6 name format illegal

```

PARAMETER ERROR: name area outside calling process.

Create peripheral process (name address, device number, result)

w0 result (return)

w1 device number (call)

w2

w3 name address (call)

jd 1<11+54

name address: process name (call and return)

Assigns a process name to a peripheral device. If the first word of the name is zero, the monitor generates a unique process name and stores it in the name area specified in the call. It is required that the calling process is a user of the device and that no other process has reserved it.

If the device is a backing storage device, already included in the bs-system by means of PREPARE BACKING STORAGE, the name must be the same as the document name.

This is a privileged function.

Base of the peripheral process is set to catalog base of calling process, except if the peripheral process is a backing storage device, in which case the base is set to (-8388607, 8388606).

result = 0 peripheral process created

1 function forbidden in calling process

2 calling process is not a user; catalog i/o error

3 name conflict; not same disc name

4 device number does not exist

5 device is reserved by another user

6 name format illegal

PARAMETER ERROR: name area outside calling process.

Create internal process (name address, parameter address, result)

w0 result (return)

w1 parameter address (call)

w2

w3 name address (call)

jd 1<11+56

name address: process name (call and return)

parameter address: (call)

+ 0 first storage address

+ 2 top storage address

+ 4 buffer claim, area claim

+ 6 internal claim, function mask

+ 8 mode (protection register, protection key)

+10 max base (lower)

+12 max base (upper)

+14 standard base = catalog base (lower)

+16 standard base = catalog base (upper)

Creates a description of an internal process with a given name and parameters. If the first word of the name is zero, the monitor generates a unique process name and stores it in the name area given in the call.

Base of the internal process name is set to the catalog base of the calling process.

The storage area must be within the storage area of the calling process. The first and top storage addresses point to the first storage word and the last storage word + 2 of the new process.

The buffer claim and area claim must be less than or equal to the claims of the calling process, which are decreased accordingly.

The internal claim must be less than the claim of the calling process, which is decreased by the value specified + 1 to include the new process itself.

In the function mask a bit equal to one indicates that the process can call privileged function with the corresponding key. Consequently the mask bits equal to one must be a subset of those defined for the calling process.

If mode is zero the new process is allowed to write in the whole primary storage (lower write limit = 8, upper write limit = core size). If mode is non-zero the new process is only allowed to write within its own corearea.

(The operating system BOSS still expects mode to correspond to the RC4000 protection register, protection key).

The new process is a child of the calling process. It is created in the state waiting for start by parent, with stop count, interrupt address, escape address, time quantum and run time equal to zero, while the instruction counter is equal to the first storage address. The interrupt mask is initialized to prevent internal interruption on arithmetic overflow.

The start time of the process is set equal to the current value of the 48-bit clock.

```

result = 0 internal process created
      1 storage area outside calling process;
        claims exceeded; illegal protection;
        max base or standard base not contained
        in corresponding base of calling process
      2 catalog i/o error
      3 name conflict
      6 name format illegal

```

Note that the internal process is not given claims on the backing store by create internal process.

PARAMETER ERROR: name area or parameter area outside calling process.

Start internal process (name address, result)

w0 result (return)

w1

w2

w3 name address (call)

jd 1<11+58

name address: process name (call)

Starts an internal process with a given name provided it is a child of the calling process in the state waiting for start by parent. Also started are all descendants of the child that were stopped previously along with it.

It is ensured that no process is started before its parent has been started. The processes are set in the running state.

Finally the stop count of the calling process is increased by one to indicate that a process, the child, is modifying its storage area.

result = 0 internal process started

2 state of process does not permit start

3 process does not exist;

process is not an internal process;

process is not a child of calling process

6 name format illegal

PARAMETER ERROR: name area outside calling process.

stop internal process (name address, message flag, buffer address, result)

w0 result (return)

w1

w2 message flag (call), buffer address (return)

w3 name address (call)

jd 1<11+60

name address: process name (call)

Initiates a stop of an internal process with a given name provided it is a child of the calling process. Also stopped are all running descendants of the child. When the stop is initiated, the child is placed in the state waiting for stop by parent, while its descendants are set to waiting for stop by ancestor. A process can be in any state when the stop is initiated. If the process is waiting for a message, an answer, an event or a process function, its state is changed to waiting for stop, as explained above, but at the same time its instruction counter is decreased by two to ensure that it will repeat the call of the corresponding monitor procedure when it is started again.

Finally, the stop procedure selects a message buffer, decreases the buffer claim of the calling process by one and returns.

The monitor completes the stop operation as follows: Each time the stop count of one of the involved processes becomes zero (after the completion of high-speed input/output), the process is transferred to the state waiting for start by ancestor (or parent) and the stop count of its parent is decreased by one.

When all involved processes are stopped, the monitor delivers an answer in the selected message buffer to the calling process, which in turn receives it by calling either wait answer or wait event.

If the buffer claim of the calling process is exceeded the buffer address is zero:

buffer address = 0 buffer claim exceeded
 > 0 selected buffer address

result = 0 stop initiated and answer will arrive,
 depending on buffer address
3 process does not exist;
 process is not an internal process;
 process is not a child of calling process
6 name format illegal

PARAMETER ERROR: name area outside calling process.

Modify internal process (name address, register address,
result)

w0 result (return)

w1 register address (call)

w2

w3 name address (call)

jd 1<11+62

name address: process name (call)

register address: (call)

+ 0 working register 0

+ 2 working register 1

+ 4 working register 2

+ 6 working register 3

+ 8 exception register

+10 instruction counter

Initializes the register values of an internal process with a
given name provided it is a child of the calling process in the
state waiting for start by parent.

result = 0 internal process modified

2 state of process does not permit modification

3 process does not exist;

process is not an internal process;

process is not a child of calling process;

6 name format illegal

PARAMETER ERROR: name area or register area outside calling
process; instruction counter outside child process.

```

remove process (name address, result)
w0 result (return)
w1
w2
w3 name address (call)
jd 1<11+64

```

```

name address: process name (call)

```

Removes an internal or external process from the monitor. Removal depends on the kind of the process:

PERIPHERAL PROCESS (except temporary links): The calling process must be a user of the peripheral process and no other process must have reserved it. In this case the name of the peripheral process is removed, but the device itself remains described within the monitor. This is a privileged function.

TEMPORARY LINKS: The calling process is removed as user of the process (and as reserver if it has reserved the process). If there is no other users (and reserver) of the process it is removed.

AREA PROCESS and ENTRY LOCK PROCESS: The calling process must be a user of the area process. In this case the calling process is removed as a user and reserver of the area process and its area claim is increased by one. If the area process has no other users, the process description is also removed.

PSEUDO PROCESS: The pseudo process must have been created by the calling process. In this case the process is removed and the area claim of the calling process is increased by one.

All message buffers are examined for pending messages sent to the pseudo process:

a. A buffer contains a message sent to, but not received by,

the pseudo process: A dummy answer (receiver does not exist) is delivered, unless the message was regretted by the sending process (in which case the buffer is merely released).

- b. A buffer contains a message sent to and received by the pseudo process:

In this case the procedure returns with result = 2.

INTERNAL PROCESS: The internal process must be a child of the calling process in the state waiting for start by parent. In this case the child and its descendants are removed as follows: the process descriptions are removed one by one starting with the youngest descendants and the buffer claims, area claims, internal claims and backing storage claims are added to those of their parents.

All area processes and entry lock processes used by the involved internal processes are removed as described above.

All pseudo processes created by the involved internal process are removed.

In all peripheral processes the internal processes are removed as users and reservers, but the peripheral processes themselves are not removed unless it is a temporary link in which case it is removed if there are no other users or reserver of the link.

Finally all message buffers are examined for pending messages or answers involving the processes to be removed. There are the following possibilities.

- a. A buffer contains a pending or received message to a removed process: A dummy answer (receiver does not exist) is delivered unless the message was regretted by the sending process. If the message was received, the buffer claim of the parent of the removed process is increased by one.

- b. A buffer contains a pending answer to a removed process:
The buffer is released and the buffer claim of the parent of the removed process is increased by one.
- c. A buffer contains a pending or received message from the removed process to another process: The message is left undisturbed with an indication that the buffer is regretted. The buffer claim of the parent of the removed process is increased by one.
- d. A buffer contains an answer from the removed process to another process: The answer is left undisturbed.

result = 0 process removed

- 1 function forbidden in calling process
- 2 state of internal process does not permit removal;
calling process is not a user of external process;
message to pseudo process has been received but not answered
- 3 described process is not a child or does not exist
- 5 peripheral process reserved by another process
- 6 name format illegal

PARAMETER ERROR: name area outside calling process.

test event (previous buffer address, next buffer address, result,
event identification)

w0 result (return)

w1 event identification (return)

w2 previous buffer address (call)

next buffer address (return)

w3

jd 1<11+66

The event queue of the calling process is scanned from previous buffer address, and a result specifying the state of next element in the queue is immediately returned. The possible states are - empty queue, next event is a message or next event is an answer (or the queue holds an immediate operation). If the next event is an answer or a message, the next buffer address and the event identification - holding the message identification or sender process description address respectively - are returned. If previous buffer address is zero, the queue is examined from the start.

The test event procedure does not remove the next buffer from the queue. This must be done by one of the procedures wait answer, wait message or get event.

If the queue of the calling process contains a message for which the calling process has already supplied a buffer (in previous call of wait or test event) the buffer supplied by the calling process is released, and the buffer claim of the calling process is increased by one. The calling process must supply a message buffer (i.e. the buffer claim of calling process is decreased by one) if next buffer address points to a message.

If the calling process has no claims and the queue holds only messages, the returned state indicates empty queue.

result = -1 empty queue; or message but claims exceeded
next buffer address:= 0

- 0 message;
 - event identification:= sender process
 - description address
- 1 answer;
 - event identification:= message identification
 - (cf. send message)
- 2 interrupt
- 4 immediate operation
 - (result 2 and 4 is only possible for a driver process).

PARAMETER ERROR: Previous buffer address does not point at a message buffer in the queue of the calling process.

```
generate name (name address, result)
w0 result (return)
w1
w2
w3 name address (call)
jd 1<11+68
```

name address: generated name (return)

Generates a name with the format:

wrk<6 digits>

followed by three NULL characters and stores it within the calling process. The six digits form an octal number modulo 8.777 777.

The generated name does not exist as a process name or a main catalog entry name.

```
result = 0 name generated
        2 catalog input/output error
```

PARAMETER ERROR: name area outside calling process.

```

copy (buffer address, first address, last address, result)
w0 result (return)
w1 first address (call), halfwords moved (return)
w2 buffer address (call)
w3 last address (call), characters moved (return)
jd 1<11+70

```

The message buffer must have been received by the calling process defining input or output. The procedure will move the corearea defined by first address and last address to or from the area defined by the buffer (according to the standard format of input/output messages) according to the operation in the buffer. The number of words moved is the smallest of:

number of words in corearea
 number of words in area described by the buffer.

```

result = 0 area copied
2 sender of buffer is stopped (w1 and w3
   undefined at return)
3 buffer describes input or output outside
   senders area; message regretted; operation
   in buffer illegal (w1 and w3 undefined at
   return)

```

PARAMETER ERROR: buffer address does not point at message buffer received by calling process; corearea outside calling process; last address < first address.

Set catalog base (name address, lower base, upper base,
return)

w0 base (lower limit of the interval) (call)

result (return)

w1 base (upper limit of the interval) (call)

w2

w3 name address (call)

jd 1<11+72

name address: process name (call)

Changes the catalog base of an internal process. The process will be the calling process when the first word of the name contains a 0 and it must be a child process otherwise. If it is a child process it must be in the state 'waiting for start by parent'. The new catalog base must be contained in the max base of the internal process and it must contain, be equal to or be contained in the standard base of the internal process.

result = 0 catalog base set

2 state of internal process does not permit
modification

3 process does not exist;

process is not an internal process;

process is not a child of calling process;

4 new base illegal

6 name format illegal

PARAMETER ERROR: name area outside calling process.

```

Set entry base (name address, lower base, upper base, result)
w0 base (lower limit of the interval) (call), result (return)
w1 base (upper limit of the interval) (call)
w2
w3 name address (call)
jd 1<11+74

```

```

name address: entry name (call)

```

The procedure will set the base of the main catalog entry specified by the name at name address, provided the entry is not protected against the calling process.

A possible corresponding auxcat entry is modified in the auxiliary catalog.

If the permkey of the entry is less than min_global_permkey the new base must be contained in - or equal to - the standard base of the calling process; otherwise the new base must be contained in - or equal to - the max base of the calling process and must contain, be equal to or be contained in the standard base of the calling process.

If the entry is used as a process, the calling process must be the only user. The name base of the process is changed to the new base.

```

result = 0 entry interval set
2 catalog i/o error: document not ready
3 entry not found; name conflict (at the new base)
4 entry protected, i.e. old base of entry name not
  contained in max base of calling process;
5 area process (or entry lock process) used by
  another process
6 name format illegal
7 maincat not present

```

PARAMETER ERROR: name area outside calling process.

lookup head and tail (name address, entry address, result)

w0 result (return)

w1 entry address (call)

w2

w3 name address (call)

jd 1<11+76

entry address: entry area (return)

name address: entry name (call)

The procedure corresponds to lookup entry, except that it moves both head and tail (17 words) to the entry area.

result = 0 entry looked up

2 catalog i/o error

3 entry not found

6 name format illegal

7 maincat not present

PARAMETER ERROR: name area or entry area outside calling process.

```

set backing storage claims (name address, document name
address, claim list address, result)
w0 result (return)
w1 claim list address (call)
w2 document name address (call)
w3 name address (call)
jd 1<11+78

```

Format of the claim list (call):

```

claim list address + 0 : entry claim (key = 0)
                  + 2 : segment claim (key = 0)
                  + 4 : entry claim (key = 1)
                  ...
claim list address * max permkey + 4;
                  + 0 : entry claim (key = max-permkey)
                  + 2 : segment claim (key = max-permkey)

```

```

document name address : document name (call)
name address          : process name (call)

```

The procedure will transfer backing storage claims, corresponding to the document specified by the document name, between the calling process and the process specified by the process name (which must be a child of the calling process).

The number (negative or positive) of entries and segments specified by the claim list are subtracted from the claims of the calling process and added to the claims of the child process.

If the number of segments specified is not an integral number of slices, it is increased to an integral number of slices.

Result 1 (= claims exceeded) is returned if either the claims of the calling process or the claims of the child would become negative by the transfer; in this case nothing is changed in the claims.

result = 0 backing storage claims set

1 claims exceeded

2 backing storage document with given name not found

3 process does not exist;

process is not an internal process;

process is not a child of calling process

6 name format (of docname or child name) illegal

PARAMETER ERROR: claim list area, document name area or process
name area outside calling process.

```

create pseudo process (name address, result)
w0 result (return)
w1
w2
w3 name address (call)
jd 1<11+80

```

name address: name (call and return)

The procedure will create a pseudo process with the given name and with the calling process as parent. The base of the process name is set to the catalog base of the calling process.

The area claim of the calling process is decreased by one. Any messages sent to the pseudo process will be linked to the queue of the parent.

```

result = 0 pseudo process created
      1 (area) claims exceeded
      2 catalog i/o error
      3 name conflict
      6 name format illegal

```

PARAMETER ERROR: name area outside calling process.

regret message (buffer address)

w0

w1

w2 buffer address (call)

w3

jd 1<11+82

The buffer claim of the calling process will be increased by one. If the buffer has not yet been received it will be released, otherwise it will be released when the receiver sends an answer.

PARAMETER ERROR: buffer address does not point at message buffer sent by calling process; the buffer defines an operation which cannot be regretted i.e. the operation is uneven.

General copy (buffer address, parameter address, result)

```

w0: result (return)
w1: parameter address (call)
    halfwords moved (return)
w2: buffer address (call)
w3:
    jd 1<11+84
parameter address: function (call)
                    first address (call)
                    last address (call)
                    start relative (call)

```

The monitor procedure, general copy, allows the receiver of a message buffer to copy information to or from a buffer area, described in the message buffer. The information may be copied in more than one step, as the relative position of the first word to be copied in the buffer area is specified by the calling process.

The message buffer described by buffer address must have been received by the calling process or by a pseudo process with the calling process as owner. The operation in the message buffer must be odd. The procedure will copy a buffer area (described by the message buffer, function and by start relative) to or from a core area described by first address and last address.

The function word has the following format:

```

00000000000000000000xxxxy
y:  0  copy from the buffer area to the core area.
    1  copy from the core area to the buffer area.
xxxx: specifies the relative position (in halfwords)
       in the message buffer of an address pair.
       the operation word in the message buffer has
       relative position 0 (zero).

```

The buffer area is defined by the pair of addresses in the message buffer and the word start relative in the parameter area.

The number of words moved is the smallest of the number of words in the core area and the buffer area.

The actual number of words is indicated through halfwords moved.

Note that all addresses are treated as if they were even.

result: 0 area described by halfwords moved has been copied.

2 sender of message buffer stopped (w1 undefined).

3 the addresses in the message buffer and in the function word points outside the sending process; message regretted; operation in message buffer not odd (w1 undefined).

PARAMETER ERROR: buffer not in queue of calling process;

corearea or parameter area outside calling process;

last address < first address; relative position of address pair in message buffer outside buffer.

look up aux entry (name address, docname address, tail address,
result)

w0 result (return)
w1 tail address (call)
w2 docname address (call)
w3 name address (call)
jd 1<11+86

name address: entry name (call)
docname address: document name (call)
tail address: number of segments (return)
+2 9 optional words (return)

Looks up an entry with a given name in the auxiliary catalog specified with the document name and copies the entry tail into the calling process.

result = 0 entry looked up
2 catalog input/output error
document not ready (or does not exist)
3 entry does not exist
6 name format illegal
7 maincat not present

PARAMETER ERROR: entry name, document name or tail area outside calling process.

Cf. look up entry, jd 1<11+42.

clear statistics in aux entry (name address, docname address,
result)

w0 result (return)

w1

w2 docname address (call)

w3 name address (call)

jd 1<11+88

docname address: document name (call)

name address: entry name (call)

Finds an entry with a given entry name, in the auxiliary catalog. If the entry describes an area the write and read access counters are cleared without changing the value of last changed. If the procedure is applied to an entry describing an area document name must correspond to the backing storage document holding the area.

result = 0	statistics initialized
2	catalog input/output error
	document not ready (or does not exist)
3	entry not found; name conflict (in auxiliary catalog)
6	name format illegal
7	maincat not present

PARAMETER ERROR: entry or document name area outside calling process.

Cf. Permanent entry in auxiliary catalog jd 1<11+90

permanent entry in auxiliary catalog (name address, doc name
address, perm key, result)

w0 result (return)

w1 perm key (call)

w2 document name address (call)

w3 name address (call)

jd 1<11+90

document name address: document name (call)

name address: entry name (call)

The procedure corresponds to the procedure permanent entry, except that an entry which does not describe an area may be inserted in the auxiliary catalog on the specified document. If the procedure is applied to an entry which describes an area, the document name in the entry and in the call must be the same.

result = 0 entry permanent

2 catalog i/o error;

document not ready (or does not exist)

3 entry not found; name conflict (in auxiliary catalog)

4 entry protected, i.e. base of entry name not contained in max base of calling process;
permkey illegal

5 area (or entry lock) process used by another process; entry is already permanent in another auxiliary catalog

6 name format illegal; claims exceeded

7 maincat not present

PARAMETER ERROR: entry name area or document area outside calling process.

```

create entry lock process (name address, result)
w0 result (return)
w1
w2
w3 name address (call)
jd 1<11+92

```

```

name address: entry name (call)
+8: name table address (return)

```

The procedure will do the same as create area process, except that the entry need not describe an area and that the name table address (cf. send message) corresponding to the process created will be stored in the word with address 'name address + 8'. The procedure is used to implement indivisibility between several accesses to the same main catalog entry.

```

result = 0 entry lock process created
1 area claims exceeded
2 catalog i/o error;
state of document does not permit this call
3 entry not found
6 name format illegal
7 maincat not present

```

PARAMETER ERROR: name area outside calling process.

```

set priority (name address, priority level, result)
w0 result (return)
w1 priority level (call)
w2
w3 name address (call)
jd 1<11+94

```

name address: process name (call)

The priority of an internal process with a given process name is set to priority level relative to the level of the calling process, provided the process is a child of the calling process. The priority levels of all descendants of this child are adjusted to obtain no changes in their relative priorities.

```

result = 0  priority changed
          3  process does not exist;
            process is not an internal process;
            process is not a child of calling process;
          6  name format illegal

```

PARAMETER ERROR: name address outside calling process, priority level < 0.

```

relocate process (name address, start address, result)
w0 result (return)
w1 start address (call)
w2
w3 name address (call)
jd 1<11+96

```

name address: process name (call)

The procedure is supposed to be used by an operating system which relocates child processes.

The internal process given by name must be a child of the calling process and the state of the child must be waiting for start by parent. The start address given must be a logical address within the logical address space of the calling process, describing the start of the core area in which the child is to run for a while.

The process description of the child is prepared for running in the core area selected, i.e.

- address base is modified to express the displacement between the physical core area in which the child was created and the core area in which the child is to run next.
- lower - and upper write limit are initialized to point at the core area selected (physical addresses).

In addition, all descendants of the child have their process descriptions modified as described above.

```

result = 0  child is ready to start in new core area.
          3  process does not exist;
            process is not a child of calling process;
            state of process does not permit this call;
          6  name format illegal

```

PARAMETER ERROR: name area outside calling process; core area
selected is outside calling process;

Change address base (name address, displacement, result)

w0 result (return)

w1 displacement (call)

w2

w3 name address (call)

jd 1<11+98

name address: process name (call)

The procedure changes the logical address space of an internal process.

The process given by name must be a child of the calling process and the child must not have any children itself. The child must be in the state waiting for start by parent. Odd displacement will be decreased by one.

The logical address space of the child is changed by adding the displacement given in call to the address base of the child and by changing all logical addresses in the process description accordingly.

```

result = 0  logical address space is changed
          1  the resulting logical address space of the
              process is not within physical address space 0
              to 8388606
          3  process does not exist;
              process is not a child of calling process;
              state of process does not permit this call;
              child process has children
          6  name format illegal
  
```

PARAMETER ERROR: name area outside calling process;

```

prepare backing storage (chainhead address, result)
w0 result (return)
w1
w2
w3 chainhead address (call)
jd 1<11+102

```

chainhead address: chainhead (call) (see subsection 3.8.1)

The procedure will include the device in the backing storage system by allocating a chaintable in the monitor, inserting the description of the auxiliary catalog (with an entrybase: (-8388607, 8388606)) in the main catalog, reserving the device for procfunc (= the part of the monitor implementing the interruptable monitor functions) and giving the calling process all claims on the new backing storage device. The chaintable in the monitor is initialized by means of the core area pointed to by w3, so that it describes only the area holding a copy of the chaintable and the auxiliary catalog. The state of the document is initialized to: AFTER PREPARE.

(The description for the auxcat must have permkey = max_permkey)

(If the maincatalog does not exist, the description of the auxcat will be inserted when the maincatalog is connected).

The calling process will be given all claims for the document:

```

entries (permkey = min_auxcat_key) = ... = entries (permkey =
max_permkey) = 15 * size of auxcat
slices (permkey = 0) = ... = slices (permkey = max_permkey) =
(<last slice number> +1) - slices for chaintable - slices for
auxcat.

```

This is a privileged function.

The device must already have the correct <documentname> and be reserved by the calling process. (This reservation is

automatically cancelled, when the reservation is taken over by procfunc).

The chain for the chainhead/chaintable must start in slice 0 !
If the chain is longer than needed, it will be cut down to the correct size. (I.e. the <last slice number> is significant).
(Maximum value of <last slice number> is 2046).

The chain for the auxiliary catalog is copied (and maybe the size of the auxiliary catalog is adjusted) as described in INSERT ENTRY. Chainkind is changed as follows: "drum" (0) is changed to "new drum" (2) and "disc" (1) is changed to "new disc" (3).

An area process is created for the auxiliary catalog and the calling process is inserted as reserver of the area process. No of keys from the chainhead is inserted in the field work1 of the area process. If no of keys is 0, size of auxcatalog is taken as no of keys.

(Notice: the calling process has the responsibility of releasing (or removing) this area process before a call of CONNECT MAIN CATALOG or INSERT BS or CREATE AUX ENTRY or REMOVE AUX ENTRY concerning this document).

```

result = 0 chaintable allocated
      1 function forbidden in calling process;
        area claims exceeded
      2 catalog i/o-error
      3 auxcat name overlap;
        auxcat name already exists
      4 document-device does not exist;
        device is not a bs-device;
        device not reserved by calling process
      5 auxcat size <= 0 or auxcat size too large;
        chainhead chain inconsistent;
        auxcat chain inconsistent
        illegal kind of chaintable;
        permkey of auxcat illegal;
        too many slices;

```

claims exceeded (too few slices for chaintable);
claims exceeded (auxcat too large);
claims exceeded (no room in maincat)
6 auxcat nameformat illegal;
docname nameformat illegal
7 no chains idle

PARAMETER ERROR: chainhead area outside calling process.

```

insert entry (chainhead address, entry address, result)
w0 result (return)
w1 entry address (call)
w2
w3 chainhead address (call)
jd 1<11+104

```

entry address: entry area (17 words) (call)

chainhead address: chainhead (call) see subsection 3.8.1.

The procedure will insert the entry in the main catalog, while reducing the bs-claims of the calling process.

Step I: The state of the document is changed to: DURING INSERT

Step II: The areaseize is compared to the number of slices used for the entry (in the chainhead/chaintable outpointed by w3):

a. The chain is too short:

The entrysize is cut down to:
slices used * slice length.

b. The chain fits the areaseize:
nothing is changed.

c. The chain is too long:

The chain is cut down to the needed number of slices.
(This situation will be normal, since the chaintable is not dumped when the areaentry is decreased).

The (adjusted) number of slice-links are copied from the sender area to the chaintable allocated for the document. If a slice-link is found already occupied in the allocated chaintable, (i.e. chain overlap), that slice-link will not be altered, but copying proceeds anyway.

The auxcat-entryclaim and slice-claims are adjusted.

Step III: If the maincatalog exists (i.e. it has already been connected), the maincat-entryclaim is adjusted.

Step IV: If the entryname (incl. base) does not conflict with any processname or any maincatalog name, the entry is inserted in the maincatalog. (If the entry is an area-entry, i.e. entrysize ≥ 0 , the documentname in the entry is changed to the documentname from the chain-head, before insertion in the maincatalog).

This is a privileged function

Notice: The bs-claims of calling process are properly adjusted, when result = 0, 3, 7.

The entry is ready for use (for example by CREATE AREA PROCESS) only when result = 0

Legal value of perm key and base of entry are:

min_auxcat_perm key $\leq$ perm key $\leq$ max perm key
 $-8388607 \leq$ lower base $\leq$ upper base ≤ 8388606

result = 0 entry inserted in main catalog

1 function forbidden in calling process
 no room in catalog (cf. 1.4.2)

2 catalog i/o error;
 document not found;
 state of document does not permit this call

3 name overlap;
 name already exists

4 calling process is not user of the device

5 permkey illegal
 interval illegal;
 chain overlap;
 chain outside limits

6 nameformat illegal;
docname format illegal;
claims exceeded
7 maincat not present

PARAMETER ERROR: entry area or chained area outside calling
process.

```
insert backing storage (docname address, result)
w0 result (return)
w1
w2 docname address (call)
w3
jd 1<11+106
```

docname address: document name (call)

The state of the document is changed to: READY.

A call of this procedure signals that all entries have been inserted in maincatalog (by means of INSERT ENTRY) and that normal use of the document may take place.

This is a privileged function.

```
result = 0 document included in bs-system
1 function forbidden in calling process
2 document not found;
state of document does not permit this call
4 calling process is not user of device
6 docname format illegal
```

PARAMETER ERROR: document name area outside calling process.

```

delete backing storage (docname address, result)
w0 result (return)
w1
w2 docname address (call)
w3
jd 1<11+108

```

docname address: document name (call)

The procedure will terminate use of the document and prepare cleaning of the main catalog.

Step I: It is checked that no internal process has areapro-
cesses belonging to the document.

Step II: It is checked that the main catalog is not situated on
the document.

Step III: The state of the document is changed to: DURING DELETE.
If the size of the catalog is less than or equal to 512
segments, then noofkeys is set to 0 and chainkind is
changed from "new drum"/"new disc" to "drum"/"disc".

This is a privileged function.

```

result = 0 document removed from bs-system
1 function forbidden in calling process
2 catalog i/o error;
   document not found
4 calling process is not user of device
5 areaprocesses exist for the document
6 main catalog on the document;
   docname format illegal

```

PARAMETER ERROR: document name area outside calling process.

2.110 Procedure Delete Entries, 110

2.110

```

delete entries (docname address, result)
w0 result (return)
w1
w2 docname address (call)
w3
jd 1<11+110

```

doc name address: document name (call)

Step I: A portion of the maincatalog is scanned and all entries belonging to the document are removed from the maincatalog and the maincat-entryclaim for the calling process is adjusted accordingly.

(Notice: all entry/slice claims corresponding to the document are undefined).

Step II: If the maincatalog is not completely scanned, result = 3 is returned. (This should be a signal to repeat the call).

Step III: When the whole maincatalog has been scanned, the chain-table is released and the device-process is removed.

This is a privileged function.

```

result = 0: all entries deleted (from main catalog) and
            chaintable released
1: function forbidden in calling process
2: catalog io-error;
   document not found;
   state of document does not permit this call
3: not all entries deleted yet
4: calling process is not user of device
6: docname format illegal

```

PARAMETER ERROR: document name area outside calling process.

```

connect main catalog (chainhead address, maincatalog name
    address, result)
w0 result (return)
w1 main catalog name address (call)
w2
w3 chainhead address (call)
jd 1<11+112

```

```

main catalog name address: main catalog name (call)
chainhead address: chainhead (call) (see 3.8.1)

```

Step I: The auxiliary catalog is searched for an entry with the specified name (and with a base = (-8388607, 8388606)). The state of the document is changed to: DURING INSERT.

Step II: The chain for the entry is copied (and maybe the area-size is changed) as described in INSERT ENTRY.

Step III: The entry is selected as main catalog, and all maincat-entryclaims are given to the calling process:

```

entries.maincatdocument (permkey = 0) = ... =
entries.maincatdocument (permkey = min_auxcat_permkey) =
    15 * size of maincat.

```

The maincat-state is set to: PRESENT. The whole main catalog area is cleared, to indicate all entries free.

Step IV: The area process for the auxiliary catalog is (re)created and the calling process is inserted as reserver of the area process. (See PREPARE BACKING STORAGE).

Step V: A description of the maincatalog and each of the existing auxiliary catalogs are inserted in the main-catalog and the maincat-entryclaim is adjusted accordingly.

This is a privileged function, controlled by both function mask bit 1 and bit 2.

Notice: legal value of permkey is: max_permkey.

```
result = 0: main catalog connected
          1: function forbidden in calling process; area claims
              exceeded
          2: catalog io-error;
              document not found;
              state of document does not permit this call
          3: name does not exist in auxcat;
              name overlap;
              name already exists
          4: calling process is not user of device
          5: maincat size <= 0;
              permkey illegal;
              interval illegal;
              chain overlap;
              chain outside limits;
          6: claims exceeded;
              docname format illegal
          7: main catalog already present
```

PARAMETER ERROR: maincatalog name area outside calling process.

2.114 Procedure Remove Main Catalog, 114

2.114

```
remove main catalog (result)
w0 result (return)
w1
w2
w3
jd 1<11+114
```

The area process for the main catalog is removed

The maincat state is set to: NOT PRESENT

This is a privileged function, controlled by both function mask bit 1 and bit 2.

result = 0: main catalog removed

7: main catalog not present

Lookup bs claims (name address, document name address, claim list address, result).

w0 result (return)
 w1 claim list address (call)
 w2 document name address (call)
 w3 name address (call)

jd 1<11+118

Name address: process name (call)

Format of claim list (return)

Claim list address	+0	entry claim	(key=0)
	+2	segment claim	(key=0)
	+4	entry claim	(key=1)
		:	
		:	
Claim list address	* max permkey + 4:		
	+0 :	entry claim	(key=max permkey)
	+2 :	segment claim	(key=max permkey)

Document name address : document name (call)

The procedure will transfer the backing storage claims corresponding to the document specified, from the internal process specified by process name, to the claim list address.

If the first word of process name is zero the call refers to the calling process.

PARAMETER ERROR: Claim list or name area outside calling process.

```

create aux entry and area process (procname address, docname
address, entry address, result)
w0 result (return)
w1 entry address (call)
w2 docname address (call)
w3 procname address (call)
jd 1<11+120

```

```

entry address: entry area (17 words) (call)
docname address: document name (call)
procname address: process name (call and return)

```

Step I: It is tested that the entryname (incl. entrybase) does not conflict with any entries in the auxiliary catalog, belonging to the document.

Step II: The state of the document is changed to: DURING AUX

Step III: The necessary slices are allocated (as in CREATE ENTRY) and the chaintable is dumped.

Step IV: The entry is inserted in the auxiliary catalog and the auxcat-entryclaim and sliceclaim are adjusted accordingly.

Step V: If the first word of procname is zero, a unique name is generated (see GENERATE NAME) and stored in the procname area specified in the call.

Step VI: An areaprocess is created with the name <procname>, for the accessing of the (newly created) area. The calling process is inserted as reserver of the areaprocess.

(Notice: base of areaprocess-name is set to catbase of calling process)

Notice: legal values of permkey and interval of entry: see INSERT ENTRY

This is a privileged function.

result = 0: entry and areaprocess created

1: function forbidden in calling process;
area claims exceeded

2: catalog io-error;
document not found;
state of document does not permit this call

3: procname overlap;
procname already exists;
entryname overlap (in auxcat);
entryname already exists (in auxcat)

4: calling process is not user of device;
claims exceeded

5: key illegal;
interval illegal

6: entryname format illegal;
procname format illegal;
docname format illegal

PARAMETER ERROR: process name area, document name area or entry
area outside calling process.

2.122 Remove Aux Entry, 122

2.122

```

remove aux entry (docname address, entry address, result)
w0 result (return)
w1 entry address (call)
w2 docname address (call)
w3
  jd<11+122

```

```

entry address: entry area (17 words) (call)
docname address: document name (call)

```

The auxiliary catalog for the specified document is searched, using the name and base of the specified entry.

If an entry exists in the auxiliary catalog, it will be removed.

Notice: The auxcat-entryclaim and sliceclaim are not adjusted, and slice-chain is not released.

(This may be of help, if two entries have overlapping chains (maybe because of hardware/software errors). In this situation a program may remove one of the entries without damaging the other one).

This is a privileged function.

```

result = 0: aux entry removed
      1: function forbidden in calling process
      2: catalog io-error;
         document not found;
         state of document does not permit this call;
      3: entry does not exist (in auxcat)
      6: entry nameformat illegal;
         docname format illegal

```

PARAMETER ERROR: document name area or entry area outside calling process.

2.124 Procedure Send Pseudo Message, 124

2.124

Send pseudo message (pseudo proc.descr, message, message flag,
name)

w0 pseudo process description (call)
w1 message address (call)
w2 message flag (call), buffer address (return)
w3 name address (call)
jdl<11+124

name address : process name (call)
+8 : name table address (return)
message address: 8 words (call)

The procedure works as SEND MESSAGE (16) with the following two exceptions:

- 1) The receiver process must be an internal process
- 2) The pseudo process description address is inserted as sender of the message buffer.

PARAMETER ERROR: Process name area or message area outside calling process; w0 does not point at a pseudo process. Pseudo process not created by calling process; receiver not an internal process.

```
set cpa (name address, cpa parameter, result)
```

```
w0 result (result)
```

```
w1 cpa (call)
```

```
w2
```

```
w3 name address (call)
```

```
jd1<11+126
```

```
name address : process name (call)
```

```
cpa : cpa value or 0 or 1 (call)
```

Sets the cpa register of an internal process.

The process must be a child of the calling process and in the state waiting for start by parent.

The process must not have any children itself. The cpa value should fulfil the following conditions:

$$8 \leq \text{cpa value} \leq \text{initial cpa value (child)}$$

If cpa = 0, the cpa value will be set to the last word of the last monitor table + 2.

If cpa = 1, the cpa value will be set equal to the initial cpa value of the child.

```
result = 0 cpa register set
```

```
2 state of process not waiting for start by parent;
  process has children
```

```
3 process does not exist;
  process not a child of calling process;
  process not an internal process
```

```
4 illegal cpa value
```

PARAMETER ERROR: Name area outside calling process.

3. IMPLEMENTATION DETAILS

3.

This chapter defines the format of queues, buffers, process descriptions and the catalog on the backing store. It also defines how these entities can be addressed within internal processes.

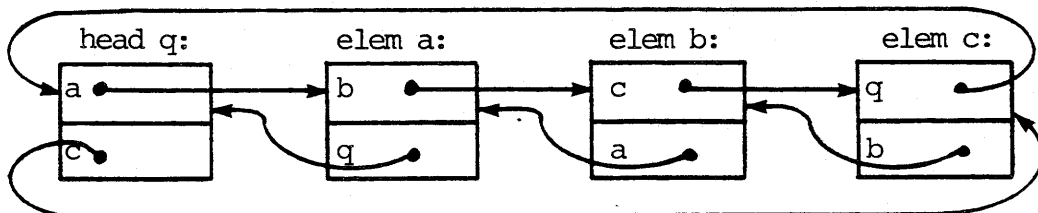
3.1 Administration of Queues

3.1

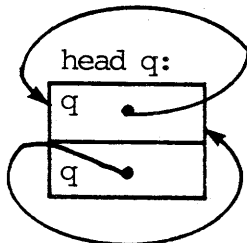
The monitor administers the following queues:

- a time slice queue
- a message buffer pool
- an event queue for each process

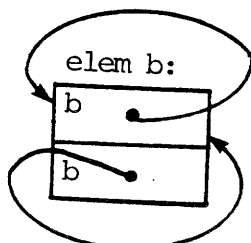
A queue consists of a head and a number of elements, which are linked cyclically. A head or element consists of two consecutive words defining the addresses of the next and previous elements in the queue, for example:



When the queue is empty, the head points to itself:



Also when an element is removed from the queue without being linked to another queue, the element points to itself:



3.2 Administration of Time Slices

3.2

The queue of running internal processes is described in the following storage locations within the monitor:

absolute address:	contents:
66	current internal process
68	next running process
70	previous running process
102	maximum time slice

CURRENT INTERNAL PROCESS is the process description address of the process executing instructions right now.

NEXT and PREVIOUS RUNNING PROCESS are the head of the time slice queue that links the process descriptions of all running processes cyclically. Note that these links are not process description addresses, but process description addresses + 18 (it points to the linkfield of the internal process, see section 3.5 Format of Internal Process Description).

The PREVIOUS RUNNING PROCESS will always point at the DUMMY PROCESS.

MAXIMUM TIME SLICE is a constant defining the maximum time quantum, in units of 0.1 millisecond, allotted to each process in turn.

3.3 Administration of Message Buffers

3.3

3.3.1 Message Buffer Pool

3.3.1

The message buffer pool is described in the following storage locations within the monitor:

absolute address:	contents:
82	Free list: first message buffer
84	- - last message buffer.

86	message pool start
88	message pool end
90	message buffer size

FREE LIST is the head of the queue of available message buffers that are linked cyclically.

MESSAGE POOL START and END are the first and last addresses of the contiguous storage area containing all message buffers.

MESSAGE BUFFER SIZE is the number of halfwords per message buffer (at present = 26 halfwords).

3.3.2 Message Buffer Format

3.3.2

A message buffer has the following format:

buffer address:	contents:
-2	message flag
+0	next buffer
+2	previous buffer
+4	receiver or answer type
+6	sender
+8	message or answer
....	
+22	message or answer

MESSAGE FLAG is a message identification (cf. SEND MESSAGE and TEST EVENT).

NEXT and PREVIOUS BUFFER are the links that connect the buffer cyclically to other buffers in its present queue.

RECEIVER and SENDER define the state of the buffer as described in subsection 3.3.3.

MESSAGE or ANSWER are eight words exchanged between two communicating processes.

Input/output messages have a standard format:

<u>message</u>	<u>answer</u>
+8 operation, mode	status word
+10 first storage address	number of halfwords
+12 last storage address	number of characters
+14 segment number	

input is signalled by: operation = 3

output is signalled by: operation = 5

3.3.3 Message Buffer States

3.3.3

The possible states of a message buffer is defined by the sign of the receiver and sender address:

<u>sender</u>	<u>receiver</u>	<u>state</u>
0	0	buffer available
+	+	message pending (i.e. buffer claim of receiver has not been decreased on account of this message)
+	-	message received (i.e. buffer claim of receiver has been decreased on account of this message)
-	+	not possible
-	-	message received, but regretted by sender

(If 'receiver' is odd the message is an immediate message which is only used in connection with driver process).

The answer type is coded as follows:

- 1 normal answer
- 2 dummy answer, message rejected
- 3 dummy answer, message unintelligible
- 4 dummy answer, receiver malfunction
- 5 dummy answer, receiver does not exist

The SENDER and RECEIVER ADDRESSES are the process description addresses of the two processes that communicate in the buffer.

In pending answers the receiver address has been replaced by the result parameter of the procedure wait answer.

The transitions between these states are clarified by the following scheme:

<u>after calling:</u>	<u>buffer state:</u>	<u>queue situation:</u>
send message	message pending	buffer in receiver queue
wait message	message received	buffer in no queue
send answer	answer pending	buffer in sender queue
wait answer	buffer available	buffer in pool
wait event	message: received	buffer in receiver queue
	answer: pending	buffer in receiver queue
get event	message: received	buffer in no queue
	answer: available	buffer in pool

3.4 Administration of Process Descriptions

3.4

The format of a process description depends on the kind of process; common to all process descriptions, however, is a head defining the kind, name and base of the process:

<u>process description address:</u>	<u>contents:</u>
-4	lower base
-2	upper base
+0	kind
+2	name
+10	other parameters

A process is considered removed if the first word of its name is zero.

The monitor has a name table containing the base addresses of all process descriptions (including those of removed processes). The

name table is described in the following storage locations:

absolute address:	contents:
72	name table start
74	first device in name table
76	first area in name table
78	first internal in name table
80	name table end

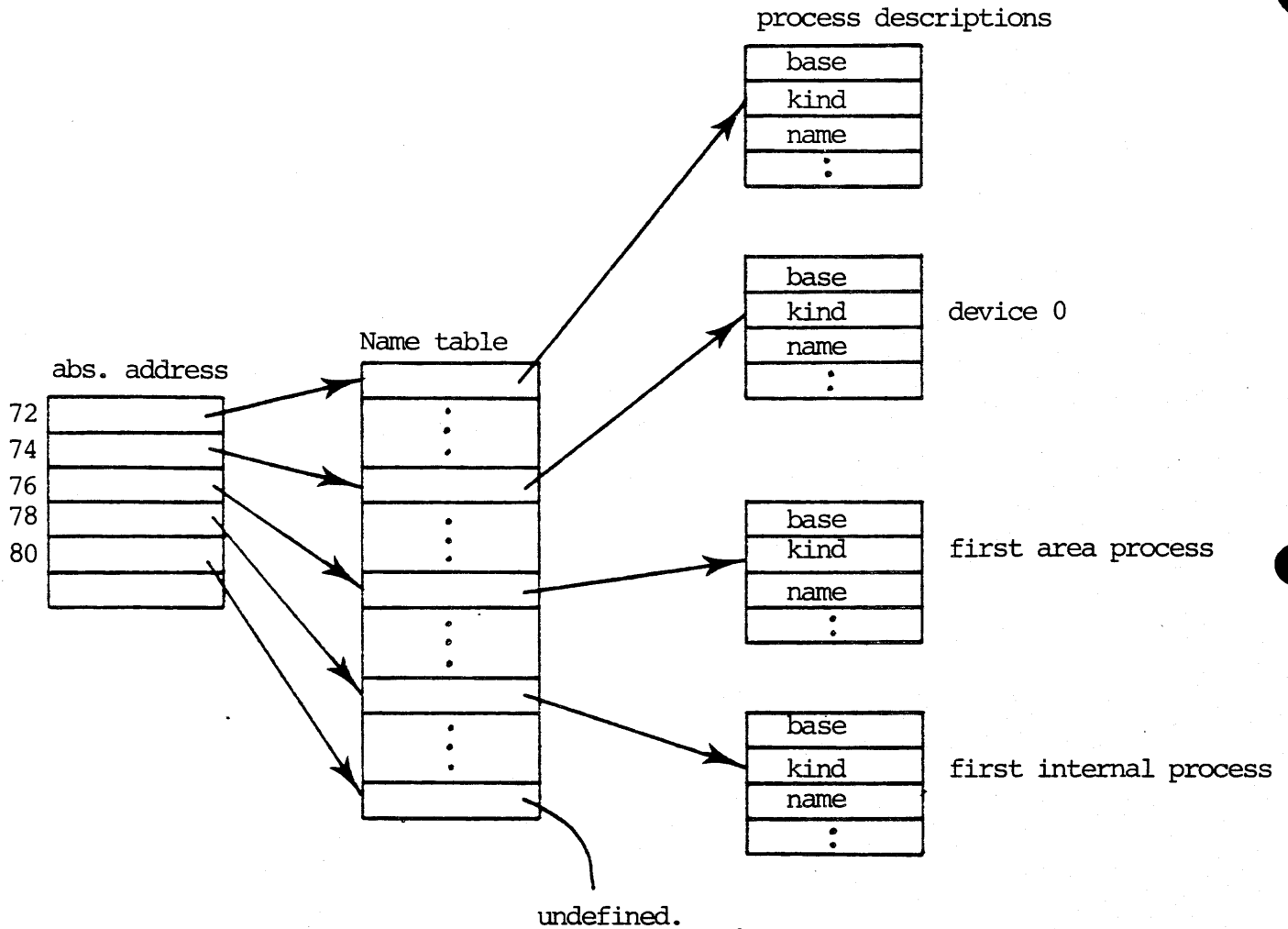
NAME TABLE START is the address of the first name table entry. This and the following name table entries contain process description addresses of special processes, used during debugging of the system and external processes corresponding to pure interrupt sources. At present the first four elements in the name-table points to the four external processes REMOTER, HOST, CLOCK and ERRORLOG.

FIRST DEVICE in name table is the address of the name table entry that contains the process description address of peripheral device number 0. The following name table entries contain process description addresses of peripheral devices 1, 2, 3, etc.

FIRST AREA in name table is the address of the name table entry that contains the process description address of the first area process. The following name table entries contain the rest of the area process description addresses.

FIRST INTERNAL in name table is the address of the name table entry that contains the process description address of the first internal process. The following name table entries contain the rest of the internal process description addresses. At present the first three elements in this part of the table contain pointers to the three processes PROCFUNC, s and DRIVERPROC.

NAME TABLE END is the address of the last name table entry. This does not contain a process description address.



3.5 Format of Internal Process Description

3.5

An internal process is described in a table with the following format:

Relative

<u>address:</u>	<u>Contents:</u>
-4	lower base of process name
-2	upper base of process name
0	kind (always 0)
+2 to +8	process name
+10, +11	stop count, process state
+12, +13	relative, id-bit
+14	event queue head: first event
+16	- - - : last event

+18	process queue link: next process
+20	- - - : previous process
+22	first address (logical)
+24	top address (logical)
+26, +27	buffer claim, area claim
+28, +29	internal claim, function mask
+30	priority
+32, +33	mode (protection register, protection key)
+34	interrupt mask
+36	user exception (interrupt) address (logical)
+38	user escape address (logical)
+40	initial cpa
+42	- base
+44	- lower write limit (physical)
+46	- top write limit (physical)
+48	- interrupt levels
+50	parent description address
+52	time quantum, unit in 0.1 msec
+54 } +56 }	run time (double word), unit in 0.1 msec
+58 } +60 }	start time (double word), unit in 0.1 msec
+62 } +64 }	start waiting time (double word), unit in 0.1 msec
+66	waiting address
+68	lower catalog base
+70	upper catalog base
+72	lower max base
+74	upper max base
+76	lower standard base
+78	upper standard base
+80	save working register 0
+82	- - register 1
+84	- - register 2
+86	- - register 3
+88	save status
+90	save ic
+92	save cause

```

+94      save sb
+96      current cpa
+98      -      base
+100     -      lower write limit
+102     -      top write limit
+104     -      interrupt levels
+106     save area address (used by driver process)
+108
:
:      save area (used by driver process)
:
+120
+122     (first process extension (co-routine monitor))
+124     (second process extension (co-routine monitor))
+126     bs claim: chain 0: key 0: entries
+128                                - 0: slices
+130                                - 1: entries
+132                                - 1: slices
+134                                - 2: entries
+136                                - 2: slices
+138                                - 3: entries
+140                                - 3: slices
+142     chain 1: - 0:
:
:
```

KIND is 0 for internal process.

NAME is a textstring of 12 ISO characters beginning with a small letter followed by a maximum of 10 small letters or digits terminated by NULL characters.

STOP COUNT defines the number of internal and external processes modifying the storage area of the present process.

PROCESS STATE has one of the following values:

```

2.0100 1000 running
2.0000 1000 running after error
2.1011 0000 waiting for stop by parent
```

2.1010 0000 waiting for stop by ancestor
 2.1011 1000 waiting for start by parent
 2.1010 1000 waiting for start by ancestor
 2.1100 1100 waiting for process function
 2.1000 1101 waiting for message
 2.1000 1110 waiting for answer
 2.1000 1111 waiting for event

The meaning of these states is defined in ref. [3].

RELATIVE, ID-BIT is the identification of the internal process.
 RELATIVE is the relative address of the user bit array element in
 an external process. It is used to check the access to external
 processes (see section 3.6).

FIRST and LAST EVENT are the link-fields of the event queue head
 that link all pending messages and answers cyclically to the pro-
 cess.

NEXT and PREVIOUS PROCESS are the elements that link the present
 process to other internal processes in the time slice queue.

FIRST and TOP ADDRESS point to the first storage word and the
 last storage word +2 of the process.

BUFFER CLAIM, AREA CLAIM and INTERNAL CLAIM define the number of
 message buffers, area process descriptions and internal process
 descriptions that are still available to the process.

FUNCTION MASK defines the privileged monitor functions that can
 be called by the process. The meaning of the mask bits is given
 in section 1.3.

PRIORITY of the internal process. Highest possible user priority
 is 0.

MODE tells whether or not the internal process is allowed to
 write in the whole primary storage (certain programs still use
 protection register and protection key as it was used in the
 RC4000 computer).

INTERRUPT MASK : the value for the interrupt mask register used during execution of the process.

EXCEPTION ADDRESS is the address of the internal interrupt procedure.

ESCAPE ADDRESS the address of the escape procedure.

INITIAL CPA, BASE, LOWER WRITE LIMIT, TOP WRITE LIMIT, INTERRUPT LEVELS contain the initial values of the corresponding registers.

PARENT DESCRIPTION ADDRESS is the process description address of the process that has created the present process.

TIME QUANTUM is the amount of execution time used by the process during the last time slice.

RUN TIME is the total execution time used by the process since its creation. (Notice: The current value of TIME QUANTUM is not included in RUN TIME).

START TIME is the value of the clock when the process was created.

WAITING TIME is the value of the clock when the process started to wait for something.

WAITING ADDRESS is a working location used by the monitor.

CATBASE, MAXBASE, STDBASE: see section 1.5, catalog protection.

SAVE WORKING REGISTERS: W0, W1, W2, W3, STATUS, IC, CAUSE, SB contains a dump of the registers when the process does not execute instructions. When the process is started the general registers are loaded from these locations.

CURRENT CPA, BASE, LOWER WRITE LIMIT, TOP WRITE LIMIT and INTERRUPT LEVELS contain the current values of the corresponding gen-

eral registers. These general registers are never dumped, but they are loaded from these locations when the process is started. They are set when the process is created and can be modified.

SAVE AREA ADDRESS and SAVE AREA are addresses of working locations only used by the driver process.

FIRST and SECOND PROCESS EXTENSION are only used in connection with the coroutine monitor.

BS CLAIMS is a table of the backing storage claims, for each logical disc (or drum). Size of the table depends on the maximum number of logical discs (and drums) in the backing storage system.

3.6 Format of External Process Description

3.6

An external process is described in a table. The format of the description depends on the kind of the external process.

Common format for all external processes is:

<u>Relative address:</u>	<u>Contents:</u>
-x	driver process description address
:	} user bit array
:	
-6, -5	
-4	lower base of process name
-2	upper base of process name
0	kind of the process
+2 to +8	name of the process
+10	other parameters

DRIVER PROCESS DESCRIPTION ADDRESS contains the address of the internal process which does the input/output in connection with this external process. The relative address of this field depends on the size of the user bit array (see below).

USER BIT ARRAY. The size of the array depends on the number of internal processes in the monitor. The size will always be an even number of halfwords and is given by:

$$Nw := (Ni + 23) // 24$$

where

Nw is number of words in the array

Ni number of internal processes.

If an internal process is user of the external process it is marked in the bit array. The bit corresponding to the "id-bit" in the internal is one in the array element (halfword) pointed out by the relative field in the internal.

LOWER BASE, UPPER BASE process name base.

KIND see section 3.12.

NAME is a textstring of 12 ISO characters beginning with a small letter followed by a maximum of 10 small letters or digits terminated by NULL characters.

3.6.1 Format of Peripheral Process Description

3.6.1

-x		driver process description address
:	}	user bit array
-6, 5		
-4	}	name base
-2		
0		kind
+2 to +3		name
+10		process description of main process
+12		reserver
+14, +15		work0, work1
+16		event queue head: first event
+18	- - -	: last event
+20		working location

PROCESS DESCRIPTION OF MAIN PROCESS, use depends on the kind of the external process (for example in a process representing a logical disc this field points to the process representing the physical disc on which the logical disc is located).

RESERVER contains the identification of an internal process (relative, id-bit) if the internal process has reserved the external process, else the field is 0.

WORK0, WORK1 are two working locations. At present they are not used in a peripheral process.

FIRST and LAST EVENT are the link-fields of the event queue head that links all messages to the process cyclically.

3.6.2 Format of Area Process Description

3.6.2

-x	driver process description address
⋮	
⋮	
-6, 5	} user bit array
-4	
-2	} name base
0	kind = 4
+2 to +8	name
+10	process description address of bs-device
(7) +12	reserver
+14, +15	work0, work1
+16	first slice of area
+18	number of segments
+20 to +26	document name
+28	number of write accesses
+30	number of read accesses

PROCESS DESCRIPTION ADDR OF BS-DEVICE:

0: just after creation of area process or after intervention
etc. on the device on which the area is located

>0: process description address of the peripheral process for the device on which the area is located.

RESERVER contains the identification of an internal process (relative, id-bit) if the internal process has reserved the process, else the field is 0.

WORK0, WORK1. If the areaprocess represents a catalog (maincatalog or aux-catalog) WORK1 contains "No of keys" for that particular catalog, else they are 0.

FIRST SLICE OF AREA contains the slicenumber of the first area-slice on the given storage device.

NUMBER OF SEGMENTS contains the size of the area in segments. If the process does not describe an area (created through monitor procedure 'create entry lock process') the size may be negative.

DOCUMENT NAME is the name of the peripheral process for the device on which the area is located.

NUMBER OF WRITE and NUMBER OF READ ACCESSES are the number of times there has been accesses to the area since the areaprocess was created.

The area process description is only used to check the validity of messages to the area.

If a message is accepted, the message is linked to the queue of the process representing the backing storage device.

3.7 Format of Pseudo Process Description

3.7

Relative addresses:

-4	lower base of process name
-2	upper base of process name
0	kind (= 64)

+2 to +8 name

+10 process description address of parent process

A pseudo process description is allocated among the area process descriptions, i.e. the name table entry of a pseudo process is found in the part of the name table defined by the contents of absolute addresses 76 and 78.

3.8 Format of a Backing Storage Document

3.8

Each bs-document contains:

1. chaintable, with information of
 - a. document name
 - b. name (and size etc.) of auxiliary catalog
 - c. slice-chains for:
 - I. the chaintable itself
 - II. the auxiliary catalog
 - III. all area-entries contained in the auxiliary catalog.
2. auxiliary catalog, with information of
 - a. names (and size etc.) of area entries, belonging to this document, with permkey $\geq$ minimum auxcat_permkey.
 - b. names etc. of non-area entries, which have been permanented (with permkey $\geq$ minimum auxcat_permkey) into this auxiliary catalog.
3. data areas.

3.8.1 Format of Chaintable, as it is on a BS-Document

3.8.1

The chaintable must occupy a contiguous set of slices, starting in slice zero.

```

+0  first slice of auxcat, bskind *8 + permkey
+2  (working location)
+4  (working location)
+6  }
+8  } name of auxiliary catalog
+10 }
+12 }
+14 size of auxcat
+16 }
+18 } documentname
+20 }
+22 }
+24 (working location)
+26 slice length
+28 last slicenumber of document, no of namekeys
+30 (working location)
+32 (working location)
+34 slice link, slice link
etc. until
+34+<last slicenumber of document>

```

The possible values for BSKIND are: 0 = drum (i.e. fast device)
 1 = disc (i.e. slow device)
 2 = new drum
 3 = new disc

The values 2 and 3 indicate that the disc-kit has been used with monitor release 9.0 or younger ones.

PERMKEY must have the value 'max permkey' (at present 3).

SLICELENGTH is the number of segments in a single slice.

LAST SLICENUMBER OF DOCUMENT must be less than or equal to 2046.

NO OF NAMEKEYS is the 'no of keys' used in this auxiliary catalog, (see subsection 1.4.2).

SLICELINK is the value to be added to a given address of a slice, to obtain the address of the next slice in a slice chain of an area. The last slice of a chain has slicelink = 0 and a free slice has slicelink = -2048.

Notice that the chaintable may contain slice-chains for area entries not described (anymore) in the auxiliary catalog.

Furthermore the chaintable may specify more slices for a specific area, than described in the auxiliary catalog. This may be due to the fact, that the chaintable is only dumped on the document when:

- a. an area entry is permanented (with permkey $\geq$ min_auxcat_permkey)
- b. an area entry (with permkey $\geq$ min_auxcat_permkey) is extended occupying new slices.

This format of a chaintable is used by the procedures prepare bs, insert entry etc. The user must read the chaintable into a buffer in his own process.

Chainhead address is then the address of the first word of the buffer.

3.8.2 Format of Chaintable, as it is in the Core-Store

3.8.2

```

-36 relative address of claims on this document in an
    internal process description
-34 first slice of auxcat, bskind*8+permkey
-32 (working location)
-30 (working location)
-28 }
-26 } name of auxiliary catalog
-24 }
-22 }
-20 size of auxcat
-18 }
```

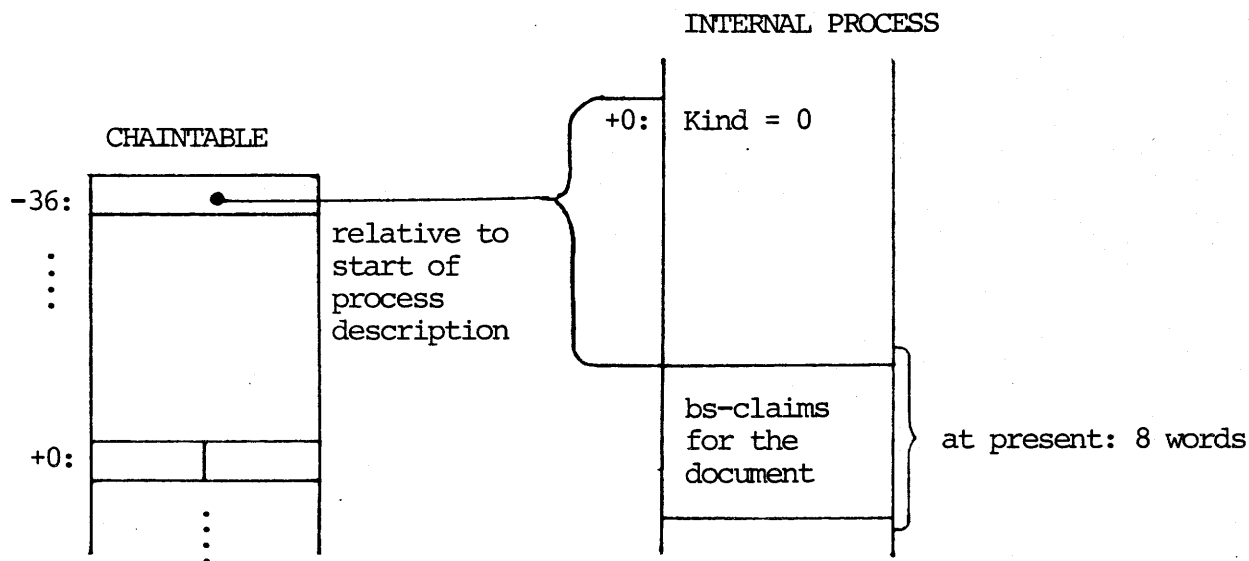
-16	}	document name
-14		
-12		
-10		(working location)
-8		slice length
-6		last slice number of document, no of namekeys
-4		state of document, (working location)
-2		(working location)
0		slice link, slice link
etc.		

Notice that this chaintable format (apart from relative address -36) resembles the format for the chaintable residing on the bs-document.

The possible states of the document is given in section 1.9.

A chaintable is considered idle when the first word of its document name is zero.

RELATIVE ADDRESS OF CLAIMS ON THIS DOCUMENT ... is shown below:



3.9 Format of a Catalog

3.9

Basically all catalogs (i.e. main catalog and auxiliary catalogs) have the same format.

A catalog is an (normal) area, accessible through an area process.

Each catalog segment contains 15 entries (of 17 words each) plus one word, the ENTRYCOUNT.

An empty catalog entry is characterized by having its first word as -1.

The entry count for a given catalog segment specifies the number of existing (i.e. non empty) entries which should be located on that segment.

3.10 Format of a Catalog Entry

3.10

HEAD:

```

+0  first slice, namekey*8+permkey
+2  lower base of entry name
+4  upper base of entry name
+6  }
+8  } entry name
+10 }
+12 }
```

TAIL:

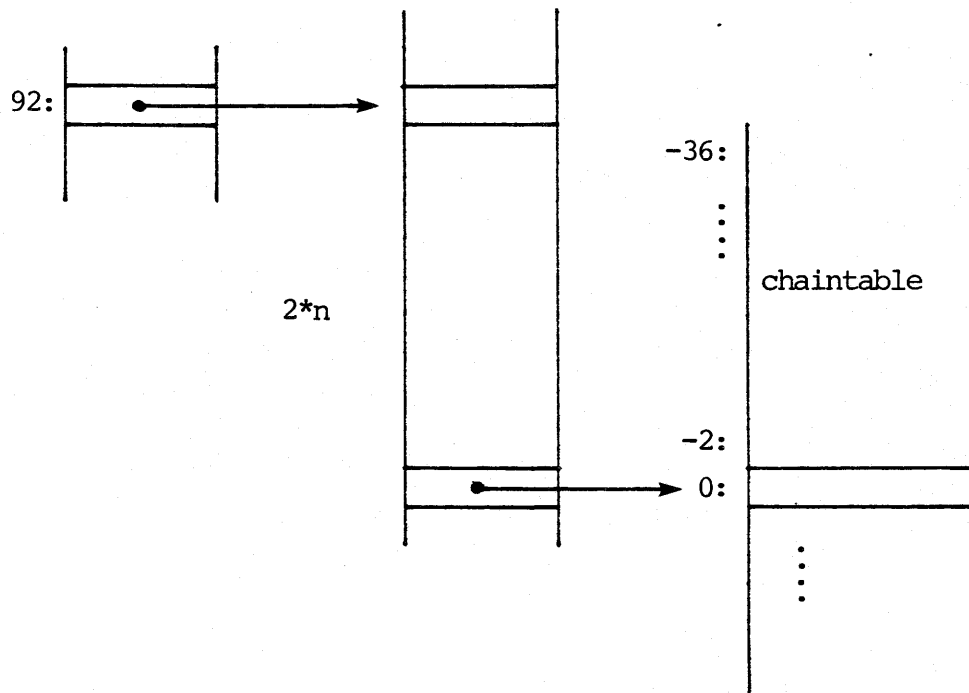
+14	size	
		<u>MAIN CATALOG</u> <u>AUX CATALOG</u>
+16	} document name or optional (see below)	entry last changed (short clock)
+18		write access counter
+20		read access counter
+22		last used
+24	}	
+26		

+28	}	optional	optional
+30			
+32			

The interpretation of a catalog entry depends on the value of the size-field (and of the permkey):

1. size >0 (independant of permkey, entry in main or auxcatalog):
 - first slice = relative address of the first slice (of the area) in the chaintable belonging to the document.
 - size = length of the area (in segments)
(Notice: need not be a multiple of slice-length)
 - documentname = name of the document on which the area belongs (entry in main catalog)
2. size =0 (independant of permkey, entry in main or auxcatalog):
 - first slice = zero
 - (size = zero)
 - documentname = name of the document, on which the area belongs (entry in main catalog).
3. size <0 and permkey < min_auxcat_permkey (entry in maincatalog):
 - first slice = zero
 - (size = optional (but less than zero))
 - (documentname = optional)
4. size <0 and permkey >= min_auxcat_permkey (entry in maincatalog):
 - first slice = $2048 + 2*n$,
where n is the chaintable number, corresponding to the document in which auxiliary catalog the entry is described.
 - (size = optional (but less than zero))
 - (document = optional)

The address of the chaintable may be found as:
 contents of (contents of (92) +2*n)



The entry which describes the maincatalog (named "catalog") contains the figure 'no of keys' in word +28.

3.11 Format of Monitor Tables

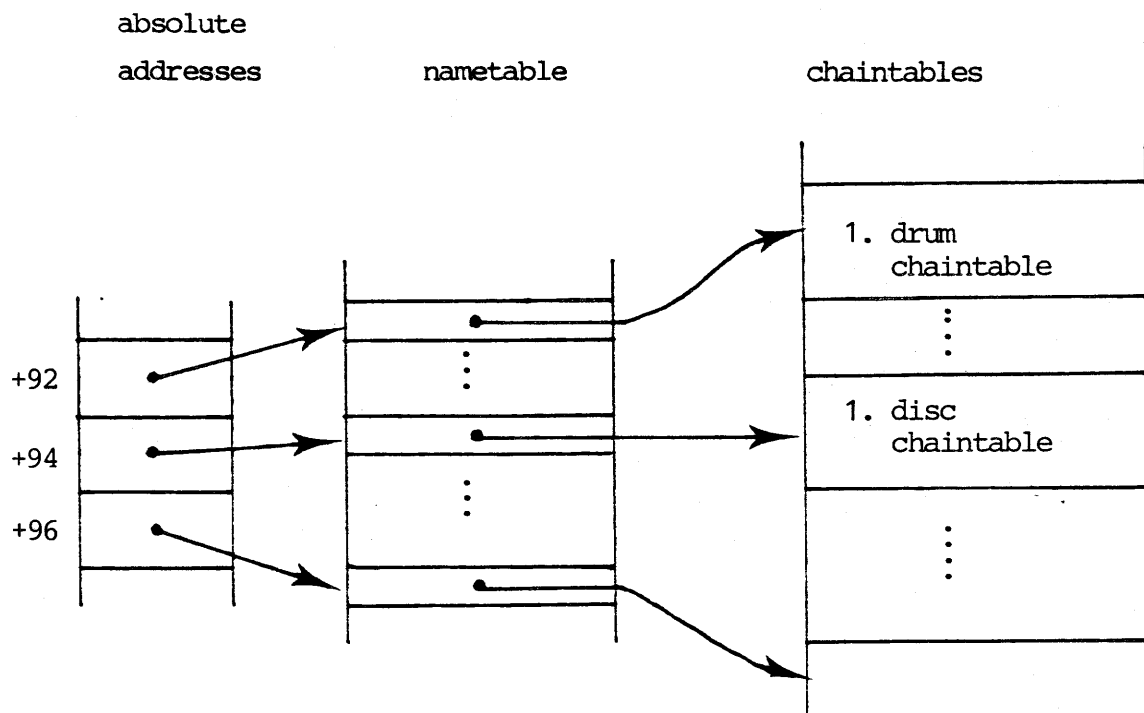
3.11

Monitor table (absolute addresses)

64	release, version of monitor
66	current internal process ?
68	next process in time slice queue
70	previous process in time slice queue
72	name table start
74	first device in name table
76	first area in name table
78	first internal in name table
80	end of process part of name table
82	free list head: first message buffer
84	- - - : last message buffer
86	first word of message pool
88	last word of message pool

90 message buffer size
 92 first drum chain in name table
 94 first disc chain in name table
 96 top chain table part of name table, i.e. the first unused entry
 98 chain table corresponding to main catalog
 100 (not used)
 102 maximum time slice
 104 time slice of current process
 106 (not used)
 108 } time in units of 0.1 msec
 110 } since dec. 31, 1967.
 112 clock value sensed
 114 (not used)
 116 number of storage halfwords
 118 min_global_permkey, min_auxcat_permkey

Layout of nametable and chaintables:



If there is no drum, 'FIRST DRUM CHAIN IN NAME TABLE' is equal to 'FIRST DISC CHAIN IN NAME TABLE'.

3.12 Kinds of External Processes

3.12

Kind Type

2	clock
4	backing store area
54	errorlog process
56	remote process
62	disc (logical or physical)
64	pseudo process
72	rc8602 (sscir)
74	rc8602 (isrpy)
76	rc8601 (sscir)
78	rc8601 (isrpy)
80	mainproc (fpa)
82	subhostproc (fpa)
84	subproc (fpa - permanent link)
85	subproc (fpa - temporary link or free)
86	receiver (fpa)
88	transmitter (fpa)
90	host

A. REFERENCES

A.

- [1] RCSL No 31-D676:
System 3 Utility Programs, Part One
- [2] RCSL No 42-i1235:
RC8000 Computer Family, Reference Manual
- [3] RCSL No 31-D476:
RC8000 Monitor, Part 1, System Design
- [4] RCSL No 31-D478:
RC8000 Monitor, Part 3, Definitions of External Processes
- [5] RCSL No 42-i1265:
BOSS 2 User's Manual

RETURN LETTER

RC8000 MONITOR, PART 2
Title: Basis System Version 2
Reference Manual

RCSL No.: 31-D697

A/S Regnecentralen af 1979/RC Computer A/S maintains a continual effort to improve the quality and usefulness of its publications. To do this effectively we need user feedback, your critical evaluation of this manual.

Please comment on this manual's completeness, accuracy, organization, usability, and readability:

Do you find errors in this manual? If so, specify by page.

How can this manual be improved?

Other comments?

Name: _____ Title: _____

Company: _____

Address: _____

Date: _____

Thank you

..... **Fold here**

..... **Do not tear - Fold here and staple**

**Affix
postage
here**

E REGNECENTRALEN
af 1979

Information Department
Lautrupbjerg 1
DK-2750 Ballerup
Denmark

E information	repl.	TSH 830824/ TSH 840705	ident	TSH 850801	page	1/1
				RC8000	class	EXT
subj. Corrections to RCSL No 31-D697: RC8000 Monitor, Part 2, Basis System Vers. 2 PN: 991 10381						
This information note contains an update of the RC8000 Monitor Part 2 manual (RCSL No 31-D697). The previous released corrections collected in RC information notes TSH 840705 and TSH 830824 have been included too. Replace the following pages: iii/iv 7/8 19/20 (Note: add 19a) 21/22 23/24 35/36 (Note: add 36a/36b) 99/100 105/106 107/108 109/110 111/112 119/120 121/122						

TABLE OF CONTENTS	PAGE
1. GENERAL MONITOR CONVENTIONS	1
1.1 Monitor Call	1
1.2 Interrupt Handling	2
1.3 Function Mask	3
1.4 Names	4
1.4.1 Names in Aux Catalog	6
1.4.2 Namekey	6
1.5 Protection and Visibility, Selection of Name	8
1.6 Permkey	10
1.6.1 Actual use of Permkey and Entry Base	11
1.7 Area Allocation, Slices	12
1.8 Strategy for Slice Allocation	13
1.9 State-Transitions and use of Document-State	14
2. DEFINITION OF MONITOR PROCEDURES	16
2.0 Set Interrupt, 0	17
2.1 Set Escape, 1	18
2.4 Process Description, 4	19
2.5 Own Process, 5	19a
2.6 Initialize Process, 6	20
2.8 Reserve Process, 8	22
2.10 Release Process, 10	24
2.12 Include User, 12	25
2.14 Exclude User, 14	26
2.16 Send Message, 16	27
2.18 Wait Answer, 18	29
2.20 Wait Message, 20	31
2.22 Send Answer, 22	32
2.24 Wait Event, 24	33
2.26 Get Event, 26	35
2.28 Test Users, Writeprotection and Reserver, 28	36
2.30 Set Writeprotect, 30	36a
2.32 Remove Writeprotect, 32	36b
2.36 Get Clock, 36	37
2.38 Set Clock, 38	38
2.40 Create Entry, 40	39
2.42 Look Up Entry, 42	41
2.44 Change Entry, 44	42
2.46 Rename Entry, 46	44
2.48 Remove Entry, 48	45

TABLE OF CONTENTS (continued)	PAGE
2.50 Permanent Entry, 50	46
2.52 Create Area Process, 52	48
2.54 Create Peripheral Process, 54	49
2.56 Create Internal Process, 56	50
2.58 Start Internal Process, 58	52
2.60 Stop Internal Process, 60	53
2.62 Modify Internal Process, 62	55
2.64 Remove Process, 64	56
2.66 Test Event, 66	59
2.68 Generate Name, 68	61
2.70 Copy, 70	62
2.72 Set Catalog Base, 72	63
2.74 Set Entry Base, 74	64
2.76 Lookup Head and Tail, 76	65
2.78 Set Backing Storage Claims, 78	66
2.80 Create Pseudo Process, 80	68
2.82 Regret Message, 82	69
2.84 General Copy, 84	70
2.86 Look up Aux Entry, 86	72
2.88 Clear Statistics in Aux Entry, 88	73
2.90 Permanent Entry in Auxiliary Catalog, 90	74
2.92 Create Entry Lock Process, 92	75
2.94 Set Priority, 94	76
2.96 Relocate Process, 96	77
2.98 Change Address Base, 98	79
2.102 Prepare Backing Storage, 102	80
2.104 Insert Entry, 104	83
2.106 Insert Backing Storage, 106	86
2.108 Delete Backing Storage, 108	87
2.110 Delete Entries, 110	88
2.112 Connect Main Catalog, 112	89
2.114 Remove Main Catalog, 114	91
2.118 Lookup bs Claims, 118	92
2.120 Create Aux Entry and Area Process, 120	93
2.122 Remove Aux Entry, 122	95
2.124 Send Pseudo Message, 124	96
2.126 Set Cpa, 126	97

```

integer word1, word2;
part_1_of_name := bit (0:47) of entry name;
part_2_of_name := bit (48:95) of entry name;
catalogsize    := size of catalog;
noofkeys       := no of keys per partition;
sum            := part_1_of_name + part_2_of_name;
word1         := sum shift (-24) extract 24;
word2         := sum extract 24; word2 := word1 + word2;
sum           := word2 + (word2 shift (-12) shift 12) // 4096;
sum           := sum shift 24 shift (-24);
segmentno     := sum mod catalogsize;
namekey       := segmentno mod noofkeys;
end;

```

In Slang the algorithm looks like:

```

dl w2 part_1_of_name
aa w2 part_2_of_name
wa w2 2
ba w2 4
al w1 0
wd w2 catalogsize
; w1 = segmentno
al w3 x1
al w2 0
wd w3 noofkeys
; w2 = namekey

```

The namekey and segment no are used when a catalog is searched for an entry with a specific name.

In a catalog with 12 segments and number of keys = 6 (the catalog is divided into two partitions) there will be the following connection between segment number and namekey:

segmentno:	0	1	2	3	4	5	6	7	8	9	10	11
namekey:	0	1	2	3	4	5	0	1	2	3	4	5

The search for an entry starts at the segment given by segmentno and continues cyclically through the catalog (last segment is followed by segment 0) until segment = segmentno + noofkeys - 1.

Only if noofkeys is equal to catalogsize overflow is allowed through the whole catalog.

If the catalog contains more than one partitions overflow is only allowed within noofkeys segments. Therefore it may occur that an entry cannot be inserted in the catalog even if there are unused entries in the catalog.

1.5. Protection and Visibility, Selection of Name.

Associated with each internal process are two PROTECTION BASES, namely the MAXBASE and the STDBASE and a VIEWPOINT BASE, namely the CATBASE.

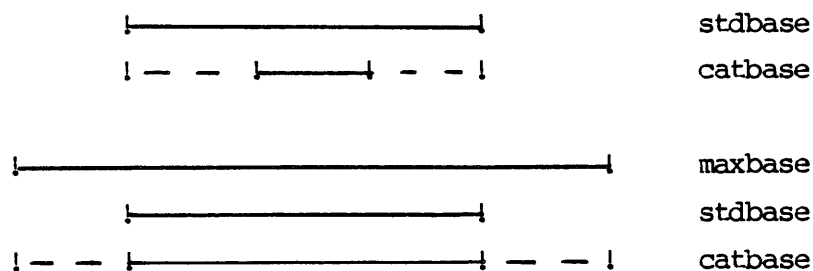
(These bases are initialized by the parent, when the internal process is created).

The catbase may freely be changed by the internal process, as long as the following is fulfilled:

lower stdbase <= lower catbase <= upper catbase <= upper stdbase
or
lower maxbase <= lower catbase <= lower stdbase
and
upper stdbase <= upper catbase <= upper maxbase,

i.e. "inside" stdbase og "between" stdbase and maxbase.

change of catalogbase:



2.4 Procedure Process Description, 4

2.4

Process description (name address, result)

w0 result (return)

w1

w2

w3 name address (call)

jd 1<11 + 4

name address: process name (call)

Checks the existence of a process with a given name. The result is the address of the process description within the monitor, if the process exists; otherwise the result is zero.

result = 0 process does not exist

> 0 process description address

PARAMETER ERROR: Process name area outside calling process.

2.5 Procedure Own Process, 5

2.5

Own process (result)

w0

w1 result (return)

w2

w3

jd 1<11 + 5

The result is the address of the process description of the calling process, within the monitor.

2.6 Procedure Initialize Process, 6

2.6

```

Initialize process (name address, result)
w0 result (return)
w1
w2
w3 name address (call)
  jd 1<11 + 6

```

name address: process name (call)

Checks the existence of a process with a given name and prepares it for communication with the calling process.

The initializing depends on the kind of the process as follows:

INTERNAL PROCESS, CLOCK, TYPEWRITER or PSEUDO PROCESS:

Only the existence of the process is checked.

PAPER TAPE READER, PAPER TAPE PUNCH:

The existence of the peripheral process with the calling process as a user is checked. The device is reserved unless it is reserved by another process.

The procedure prepares the peripheral process so that input/output in flexo-mode is assumed to be in lower case.

OTHER PERIPHERAL PROCESS:

The existence of the peripheral process with the calling process as a user is checked. If the peripheral process controls a sequential document, (e.g. card reader, magnetic tape) it is reserved for the calling process unless another process has reserved it. Further details, see ref. 4.

If the external process represents a temporary link the calling process is included as user of the process.

AREA PROCESS:

The existence of the area process is checked. If the calling process is not a user of the area process, result 3 is returned. After initialization, only input from the area is allowed, unless the process has already been reserved by the internal process.

result = 0 process initialized

1 reserved by another process

2 calling process is not a user

3 process does not exist

calling process is not a user of area process.

PARAMETER ERROR: process name area outside calling process.

2.8 Procedure Reserve Process, 8

2.8

```

reserve process (name address, result)
w0 result (return)
w1
w2
w3 name address (call)
jd l<ll + 8

```

name address: process name (call)

Reserves a process with a given name for exclusive communication with the calling process.

Reservation depends on the kind of the process as follows:

INTERNAL PROCESS, CLOCK or PSEUDO PROCESS:

Reservation is not allowed.

TYPEWRITER:

The existence of the peripheral process with the calling process as user is checked. It is reserved for the calling process unless another process has reserved it. When the typewriter is reserved, the 'att' request will be suppressed at succeeding pushes of the attention key and the attention-message will be sent directly to the reserver.

PAPER TAPE READER, PAPER TAPE PUNCH:

The existence of the peripheral process with the calling process as a user is checked. The device is reserved unless it is reserved by another process.

The procedure prepares the peripheral process so that input/output in flexo-mode is assumed to be in lower case.

OTHER PERIPHERAL PROCESS:

The existence of the peripheral process with the calling process as a user is checked. It is reserved for the calling process unless another process has reserved it. Further details, see ref. 4.

AREA PROCESS:

The existence of the area process is checked. If the calling process is not a user of the area process, result 3 is returned. The area process is reserved for the calling process provided no other process has reserved or writeprotected it, and provided the area is not protected against the calling process, see section 1.5. After reservation both input and output to the area are allowed.

result = 0 process reserved

= 1 reserved by another process

= 2 calling process is not a user
process cannot be reserved

= 3 process does not exist

calling process is not a user of area process

= 4 area process is writeprotected by other(s).

PARAMETER ERROR: process name area outside calling process.

2.10 Procedure Release Process, 10

2.10

```
release process (name address)
```

```
w0
```

```
w1
```

```
w2
```

```
w3 name address (call)
```

```
jd 1<11 + 10
```

```
name address: process name (call)
```

Releases a possible reservation of a process with a given name.

Release depends on the kind of the process as follows:

INTERNAL PROCESS, PSEUDO PROCESS:

Nothing is done.

EXTERNAL PROCESS:

If the external process exists and is reserved by the calling process, the reservation is cancelled.

PARAMETER ERROR: process name area outside calling process.

This page is intentionally left blank.

2.26 Procedure Get Event, 26

2.26

```
get event (buffer address)
w0
w1
w2 buffer address (call)
w3
jd 1<11 + 26
```

Removes a given buffer from the queue of the calling process. If the buffer contains a message, it is made ready for the sending of an answer. In this case the calling process must supply a message buffer (i.e. the buffer claim of the calling process is decreased by one) if it has not already done so (see wait message section 2.10). If the buffer contains an answer, it is released and the buffer claim of the calling process is increased by one.

The copying of information from the buffer (sender description address, message or answer and result) must be done by the calling process itself before get event is called. This requires a knowledge of the format of a message buffer (see subsection 3.3.2).

PARAMETER ERROR: buffer address does not point at a message buffer in the queue of the calling process.

2.28 Procedure Test Users, Writeprotection and Reserver, 28

2.28

Test users, writeprotection and reserver (internal name address,
external name address, answer, result)

```
w0 result (return)
w1 internal name address (call)
w2 answer (return)
w3 external name address (call)
jd 1<11 + 28
```

Internal name address: name of internal process (call)

External name address: name of external process (call)

Test whether an internal process is user or reserver of an external process, and if there are other user or another reserver.

If the first word of the name of the internal process is zero, the call refers to the calling process.

If the external process is an area process it is tested whether the internal process or other internal processes has writeprotected the area.

The success of the call is signalled in result where

```
result = 0 ok, further information in w2
        3 internal process does not exist
        4 external process does not exist
```

The answer is signalled in w2 in bit 18-23, where

```
bit 23 = 1 internal process is user
bit 22 = 1 internal process is reserver
bit 21 = 1 other user(s)
bit 20 = 1 another process is reserver
bit 19 = 1 internal process has writeprotected
bit 18 = 1 other(s) has writeprotected
```

2.30 Procedure Set Writeprotect, 30

2.30

Set writeprotect (external name address, result)

w0 result (return)

w1

w2

w3 external name address (call)

jd 1<11 + 30

external name address: name of area process (call).

Writeprotects an area process with a given name.

Only area processes can be writeprotected.

When an area process has been writeprotected it is not possible for other internal processes to reserve the area process, but they can still read from the area.

An area process can only be writeprotected if the name base of the area is within (or equal to) the max base of the calling process (i.e. it is possible to reserve the area process).

The calling process must be user of the area process.

There must be no reserver except the calling process.

Result = 0 area writeprotected

1 area reserved by other

2 not possible to writeprotect

3 calling process is not user of area process

process unknown

process is not an area

2.32 Procedure Remove Writeprotect, 32

2.32

Remove writeprotect (external name address, result)

w0 result (return)

w1

w2

w3 external name address (call)

jd 1<11 + 32

external name address: name of area process (call)

Removes a writeprotection (previously done by the calling process) of a process with a given name.

It is only possible to remove a write protection of an area.

result = 0 writeprotection removed

3 process unknown

process is not an area

3.2 Administration of Time Slices

3.2

The queue of running internal processes is described in the following storage locations within the monitor:

absolute address:	contents:
68	next running process
70	previous running process
102	maximum time slice

NEXT and PREVIOUS RUNNING PROCESS are the head of the time slice queue that links the process descriptions of all processes which is going to run. Note that these links are not process description addresses, but process description addresses + 18 (it points to the linkfield of the internal process, see section 3.5 Format of internal process description).

MAXIMUM TIME SLICE is a constant defining the maximum time quantum, in units of 0.1 millisecond, allotted to each process in turn.

3.3 Administration of Message Buffers

3.3

3.3.1 Message Buffer Pool

3.3.1

The message buffer pool is described in the following storage locations within the monitor:

absolute address:	contents:
82	free list: first message buffer
84	- - last message buffer
86	message pool start
88	message pool end
90	message buffer size

FREE LIST is the head of the queue of available message buffers that are linked cyclically.

MESSAGE POOL START and END are the first and last addresses of the contiguous storage area containing all message buffers.

Message pool start points at word -4 relative to the message buffer address of the first message in the pool.

MESSAGE BUFFER SIZE is the number of halfwords per message buffer (at present = 28 halfwords).

3.3.2 Message Buffer Format

3.3.2

A message buffer has the following format:

buffer address:	contents:
- 4	internal message state
- 2	message flag
0	next buffer
+ 2	previous buffer
+ 4	receiver or answer type
+ 6	sender
+ 8	message or answer
.....	
+22	message or answer

INTERNAL MESSAGE STATE is used by drivers for devices connected through 'intelligent' device controllers (IDA801, IFP801).

MESSAGE FLAG is a message identification (cf. SEND MESSAGE and TEST EVENT).

NEXT and PREVIOUS BUFFER are the links that connect the buffer cyclically to other buffers in its present queue.

RECEIVER and SENDER define the state of the buffer as described in subsection 3.3.3.

MESSAGE or ANSWER are eight words exchanged between two communicating processes.

+18	process queue link: next process
+20	- - - : previous process
+22	first address (logical)
+24	top address (logical)
+26, +27	buffer claim, area claim
+28, +29	internal claim, function mask
+30	priority
+32, +33	mode (protection register, protection key)
+34	interrupt mask
+36	user exception (interrupt) address (logical)
+38	user escape address (logical)
+40	initial cpa
+42	- base
+44	- lower write limit (physical)
+46	- top write limit (physical)
+48, +49	- interrupt levels (disable, enable)
+50	parent description address
+52	time quantum, unit in 0.1 msec
+54	} run time (double word), unit in 0.1 msec
+56	
+58	} start time (double word), unit in 0.1 msec
+60	
+62	} start waiting time (double word), unit in 0.1 msec
+64	
+66	waiting address
+68	lower catalog base
+70	upper catalog base
+72	lower max base
+74	upper max base
+76	lower standard base
+78	upper standard base
+80	save working register 0
+82	- - register 1
+84	- - register 2
+86	- - register 3
+88	save status
+90	save ic
+92	save cause
+94	save sb

```

+96      current cpa
+98      -      base
+100     -      lower write limit
+102     -      top write limit
+104     -      interrupt levels
+106     pu-index
+108     save area address (used by driver process)
+110
:
:      save area
:
+120
+122     (first process extension (coroutine monitor))
+124     (second process extension (coroutine monitor))
+126     bs claim: chain 0:  key 0:  entries
+128                -  0:  slices
+130                -  1:  entries
+132                -  1:  slices
+134                -  2:  entries
+136                -  2:  slices
+138                -  3:  entries
+140                -  3:  slices
+142                chain 1: -  0:
:
:
```

KIND is 0 for internal process.

NAME is a textstring of 12 ISO characters beginning with a small letter followed by a maximum of 10 small letters or digits terminated by NULL characters.

STOP COUNT defines the number of internal and external processes modifying the storage area of the present process.

PROCESS STATE has one of the following values:

```

2.0000 1011 running
2.1100 1000 waiting for cpu
2.0000 1000 running after error
2.1011 0000 waiting for stop by parent
```

2.1010 0000 waiting for stop by ancestor
 2.1011 1000 waiting for start by parent
 2.1010 1000 waiting for start by ancestor
 2.1100 1100 waiting for process function
 2.1000 1101 waiting for message
 2.1000 1110 waiting for answer
 2.1000 1111 waiting for event

The meaning of these states is defined in ref. 3.

RELATIVE, ID-BIT is the identification of the internal process.
 RELATIVE is the relative address of the user bit array element in an external process. It is used to check the access to external processes (see section 3.6).

FIRST and LAST EVENT are the link-fields of the event queue head that link all pending messages and answers cyclically to the process.

NEXT and PREVIOUS PROCESS are the elements that link the present process to other internal processes in the time slice queue.

FIRST and TOP ADDRESS point to the first storage word and the last storage word +2 of the process.

BUFFER CLAIM, AREA CLAIM and INTERNAL CLAIM define the number of message buffers, area process descriptions and internal process descriptions that are still available to the process.

FUNCTION MASK defines the privileged monitor functions that can be called by the process. The meaning of the mask bits is given in section 1.3.

PRIORITY of the internal process. Highest possible user priority is 0.

MODE tells whether or not the internal process is allowed to write in the whole primary storage (certain programs still use protection register and protection key as it was used in the RC4000 computer).

INTERRUPT MASK is the value for the interrupt mask register used during execution of the process.

EXCEPTION ADDRESS is the address of the internal interrupt procedure.

ESCAPE ADDRESS is the address of the escape procedure.

INITIAL CPA, BASE, LOWER WRITE LIMIT, TOP WRITE LIMIT, INTERRUPT LEVELS contain the initial values of the corresponding registers.

PARENT DESCRIPTION ADDRESS is the process description address of the process that has created the present process.

TIME QUANTUM is the amount of execution time used by the process during the last time slice. Note that this is undefined when the process is running.

RUN TIME is the total execution time used by the process since its creation. (notice: The current value of TIME QUANTUM is not included in RUN TIME).

START TIME is the value of the clock when the process was created.

WAITING TIME is the value of the clock when the process started to wait for something.

WAITING ADDRESS is a working location used by the monitor.

CATBASE, MAXBASE, STDBASE: see section 1.5, catalog protection.

SAVE WORKING REG. W0, W1, W2, W3, STATUS, IC, CAUSE, SB contains a dump of the registers when the process does not execute instruction. When the process is started the general registers are loaded from these locations.

CURRENT CPA, BASE, LOWER WRITE LIMIT, TOP WRITE LIMIT and INTERRUPT LEVELS contains the current values of the corresponding general registers. These general registers are never dumped, but they are loaded from these locations when the process is started. They are set when the process is created and can be modified.

PU-INDEX is the index of the cpu on which the process was running last time. This is only defined in a multiprocessor system.

SAVE AREA ADDRESS and SAVE AREA are addresses of working location used by the system.

FIRST and SECOND PROCESS EXTENSION are only used in connection with the coroutine monitor.

BS CLAIMS is a table of the backing storage claims, for each logical disc (or drum). Size of the table depends on the maximum number of logical discs (and drums) in the backing storage system.

3.6 Format of External Process Description

3.6

An external process is described in a table. The format of the description depends on the kind of the external process.

Common format for all external processes is:

Relative address:	Contents:
-x	driver process description address
:	} user bit array
-6, -5	
-4	
-2	lower base of process name
0	upper base of process name
+2 to +8	kind of the process
+10	name of the process
	other parameters

DRIVER PROCESS DESCRIPTION ADDRESS contains the address of the internal process which does the input/output in connection with this external process. The relative address of this field depends on the size of the user bit array (see below). If the external process represents a device which is connected through an IDA or IFP controller, driver process description address equals -1, as the drivers for these devices are a part of the monitor.

USER BIT ARRAY. The size of the array depends on the number of internal processes in the monitor. The size will always be an even number of halfwords and is given by:

$$N_w := (N_i + 23) // 24$$

where

N_w is number of words in the array

N_i number of internal processes.

If an internal process is user of the external process it is marked in the bit array. The bit corresponding to the "id-bit" in the internal is one in the array element (halfword) pointed out by the relative field in the internal.

LOWER BASE, UPPER BASE process name base.

KIND see section 3.12.

NAME is a textstring of 12 ISO characters beginning with a small letter followed by a maximum of 10 small letters or digits terminated by NULL characters.

3.6.1 Format of Peripheral Process Description

3.6.1

```
-x      driver process description address
:
-6, -5 } user bit array
-4      }
-2      } name base
0
+2 to +8 name
+10     process description of main process
+12     reserver
+14, +15 work0, work1
+16     event queue head: first event
+18     - - - : last event
+20     working location
```

PROCESS DESCRIPTION OF MAIN PROCESS, use depends on the kind of the external process (for example in a process representing a logical disc this field points to the process representing the physical disc on which the logical disc is located).

RESERVER contains the identification of an internal process (relative, id-bit) if the internal process has reserved the external process, else the field is 0.

WORK0, WORK1 are two working locations. At present work0 is used in a peripheral process representing a device connected through an intelligent device controller.

FIRST and LAST EVENT are the link-fields of the event queue head that links all messages to the process cyclically.

3.6.2 Format of Area Process Description

3.6.2

-y	}	writeprotection bit array
:		
-x		driver process description address
:	}	user bit array
-6, -5		
-4	}	name base
-2		
0		kind = 4
+2 to +8 name		
+10		process description address of bs-device
+12		reserver
+14, +15 work0, work1		
+16		first slice of area
+18		number of segments
+20 -+26 document name		
+28		number of write accesses
+30		number of read accesses

WRITE PROTECTION BIT ARRAY. The size and use of this array corresponds to the size and use of the user bit array, except that it concerns writeprotection of the area.

PROCESS DESCRIPTION ADDRESS OF BS-DEVICE:

- 0 just after creation of area process or after intervention etc. on the device on which the area is located (never zero in an area process representing an area located on an IDA-disc).
- >0 process description address of the peripheral process for the device on which the area is located.

RESERVER contains the identification of an internal process (relative, id-bit) if the internal process has reserved the process, else the field is 0.

WORK0, WORK1. If the area process represents a catalog (maincatalog or aux-catalog) WORK1 contains "no of keys" for that particular catalog, else they are 0.

FIRST SLICE OF AREA contains the slicenumber of the first areaslice on the given storage device.

NUMBER OF SEGMENTS contains the size of the area in segments. If the process does not describe an area (created through monitor procedure 'create entry lock process') the size may be negative.

DOCUMENT NAME is the name of the peripheral process for the device on which the area is located.

NUMBER OF WRITE and NUMBER OF READ ACCESSES are the number of times there has been accesses to the area since the areaprocess was created.

The area process description is only used to check the validity of messages to the area.

If a message is accepted, the message is linked to the queue of the process representing the backing storage device.

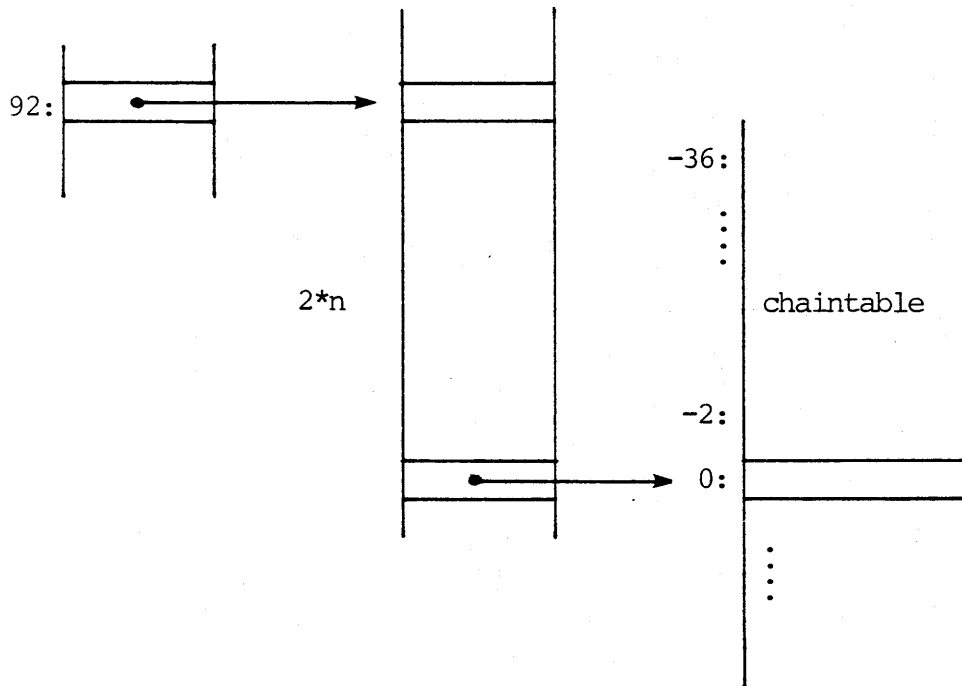
3.7 Format of Pseudo Process Description

3.7

Relative addresses:

- 4 lower base of process name
- 2 upper base of process name
- 0 kind (=64)

The address of the chaintable may be found as:
 contents of (contents of (92) + 2 * n)



The entry which describes the maincatalog (named "catalog") contains the figure 'no of keys' in word +28.

3.11 Format of Monitor Tables

3.11

Monitor table (absolute addresses)

64 release, version of monitor
 66 undefined
 68 next process in time slice queue
 70 previous process in time slice queue
 72 name table start
 74 first device in nametable
 76 first area in name table
 78 first internal in name table
 80 end of process part of name table
 82 free list head: first message buffer
 84 - - - : last message buffer
 86 first word of message pool
 88 last word of message pool

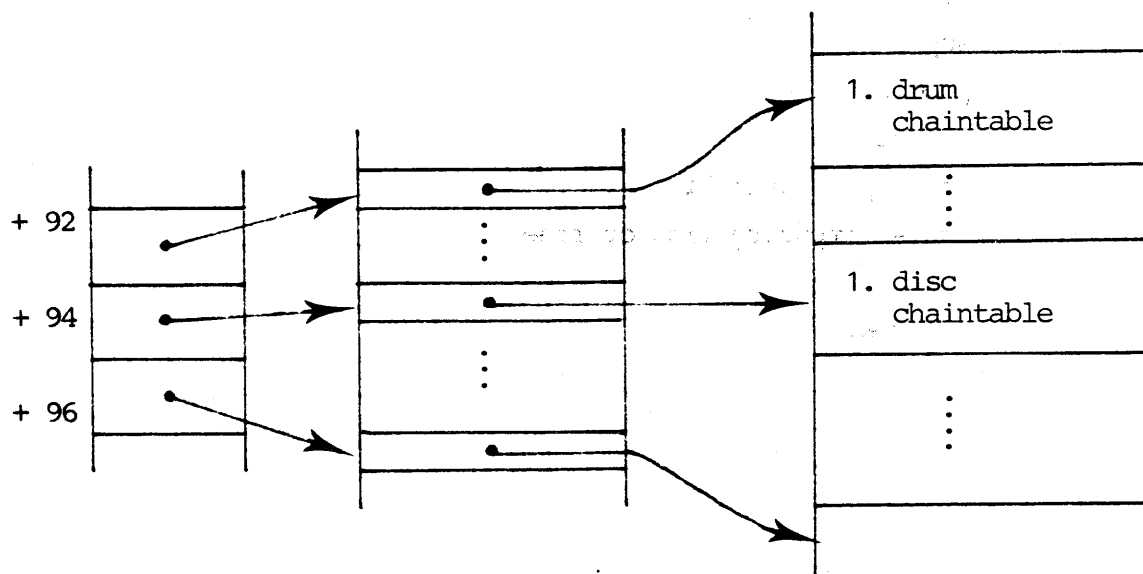
90 message buffer size
 92 first drum chain in name table
 94 first disc chain in name table
 96 top chain table part of name table, i.e. the first unused entry
 98 chain table corresponding to main catalog
 100 undefined
 102 maximum time slice
 104 time slice of current process
 106 (not used)
 108 }
 110 } time in units of 0.1 msec since dec. 31, 1967.
 112 clock value sensed
 114 (not used)
 116 number of storage halfwords
 118 min_global_permkey, main_auxcat_permkey

Layout of nametable and chaintables:

absolute
addresses

nametable

chaintable



If there is no drum, 'FIRST DRUM CHAIN IN NAME TABLE' is equal to 'FIRST DISC CHAIN IN NAME TABLE'.

3.12 Kinds of External Processes

3.12

Kind	Type
2	clock
4	backing store area
6	disc (IDA, physical - logical)
18	magnetic tape (IDA)
20	IDA main process
26	IFP main process
28	general sequential process (adp)
54	errorlog process
56	remoter process
62	disc (physical - logical)
64	pseudo process
68	free IDA/IFP sub-process
72	rc8602 (sscir)
74	rc8602 (isrpy)
76	rc8601 (sscir)
78	rc8601 (isrpy)
80	FPA main process
82	subhostproc (fpa)
84	subproc (fpa - permanent link)
85	subproc (fpa - temporary link or free)
86	receiver (fpa)
88	transmitter (fpa)
90	host

