Title:

DOMUS, System Programmer's Guide, version 3.

A/S REGNECENTRALEN

RC SYSTEM LIBRARY: FALKONERALLE 1 DK-2000 COPENHAGEN F

RCSL No: 43-GL 8374
Edition: January 1979
Author: Knud Henningsen

Keywords: MUS, Operating System, Loader, Disc, Version 3.

Abstract: This manual describes the interface between assembly program and DOMUS.

(32 printed pages)

CONTENTS PAGE

1. INTRODUCTION. 1.

The DOMUS system consists of the following software components:

Monitor
Utility Procedures
Basic I/O
Character I/O
Record I/O
Paging System
Disc Driver
Teletype Driver
File Management System
Operating System S

The operating system S takes care of core storage management, program/process load from disc files, program/process removal. It executes commands keyed in from the teletype or sends to the process from another process in the system. It introduces a process hierachy among the processes defining a parent/child relation between processes.

2. CORE STORAGE STRUCTURE.

2.1 Core Items.

Programs and processes are organized as described in [1]. After system initialization the available core storage above the basic system is organized as one large item of core storage. When the system works, pieces of that core storage is occupied by additional programs, procedures, processes, or data. Core storage is allocated in disjoint pieces, called core items, all chained together in ascending order in the one and same chain, the core item chain, the head of which being addressed by the page zero location: CORE. The allocation strategy is given in app.F.

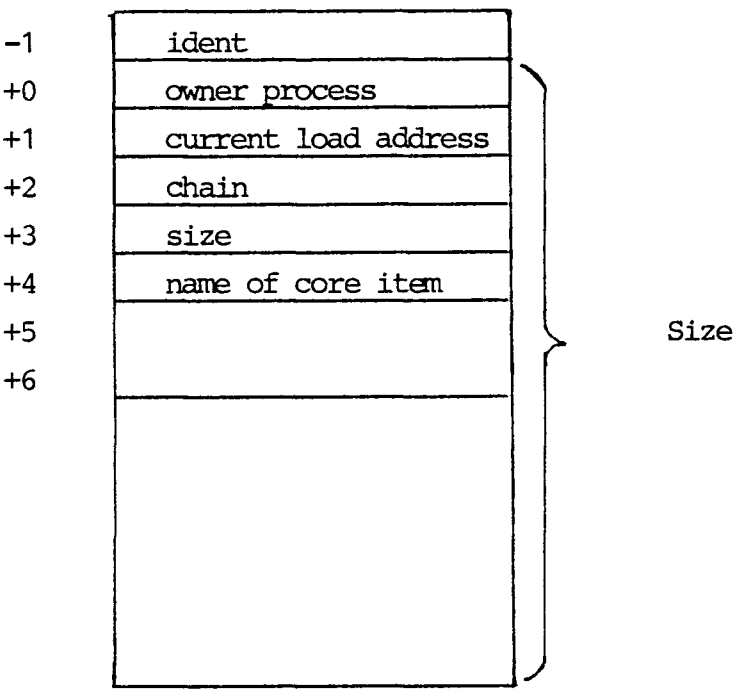
All processes except for the processes in the basic system are contained in exactly one core item. In a core item may reside several processes. Thus two relations exists between a process p and a core item C.

1) p in C =
   C contains p:
   — p lies inside the core item C

2) p owns C =
   C owned p:
   — C's owner is p

2.1.1 The Core Item Head. 2.1.1

The core item is headed by 8 word descriptor with the following contents:



The ident field is used to distinguish between the two types of items, the N-item and the X-item. The N-item is restricted for use in allocating low core storage area only, address room (0;32 Kw). The X-item can be used in any core-environment with address room (0;64 Kw). Both a N-item and an X-item could be allocated in one task, in which case they are referred to as a composite item, with the X-item head pointed out from the ident field of the N-item. In the following the term core item is used for a composite item too.

	FREE	USED
N-item:	ident:=0	ident:=0/corresp.x-item. address
X-item:	ident:=-1	ident:=-1

The owner process field contains the process description address of the process which allocated the item.

The current load address field points to the first address to be used if the core item is loaded with a procedure or a process.

The chain field points to the next core item or is set to zero if it is the last item in the chain.

The size field contains the size of the core item.

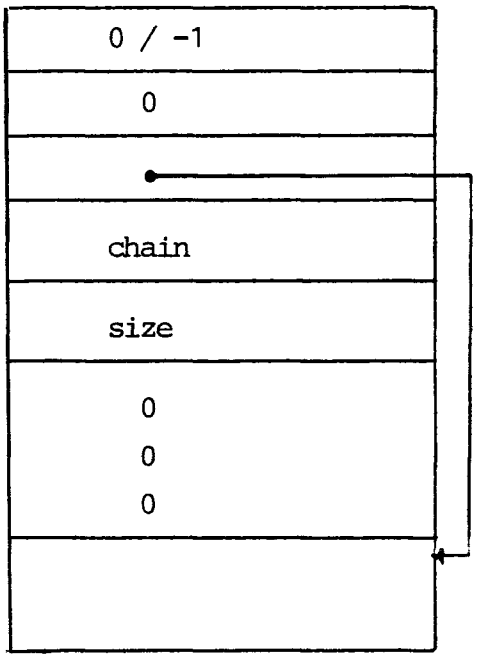
The name field contains a possible name of the core item.

Three different kinds of core items exists:

The Free Core Item
The Used Core Item
The Utility Core Item.

2.1.2 The Free Core Item. 2.1.2

The item is not used by any process.

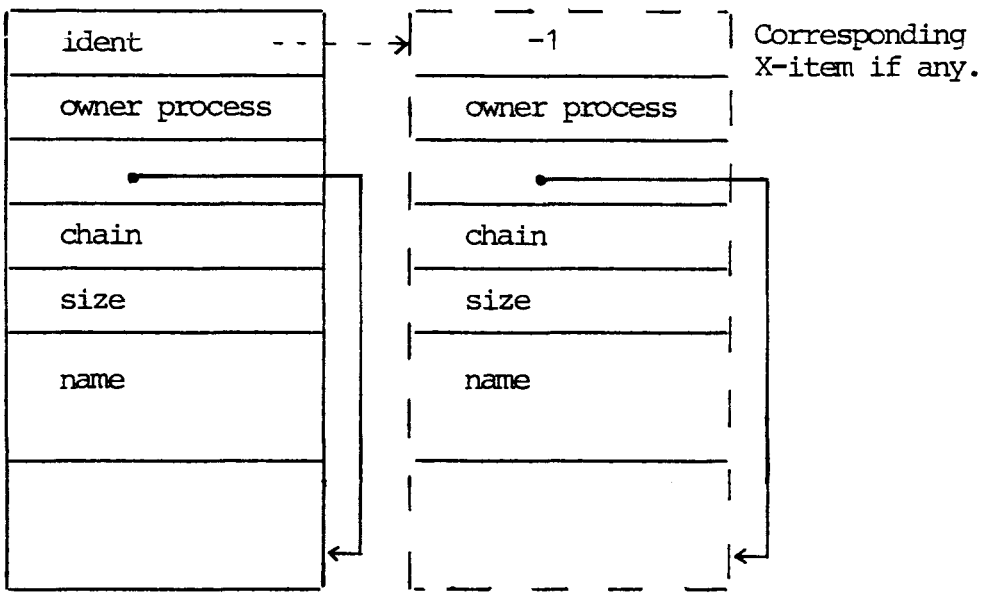


0: N-item;
-1: X-item;

Notice: a free item is never composite. A free item with the item head address in lower core is defined as a N-item. In all other cases the whole item should rest in lower core to be called a N-item.

2.1.3 The Used Core Item.

The item is allocated by a process.

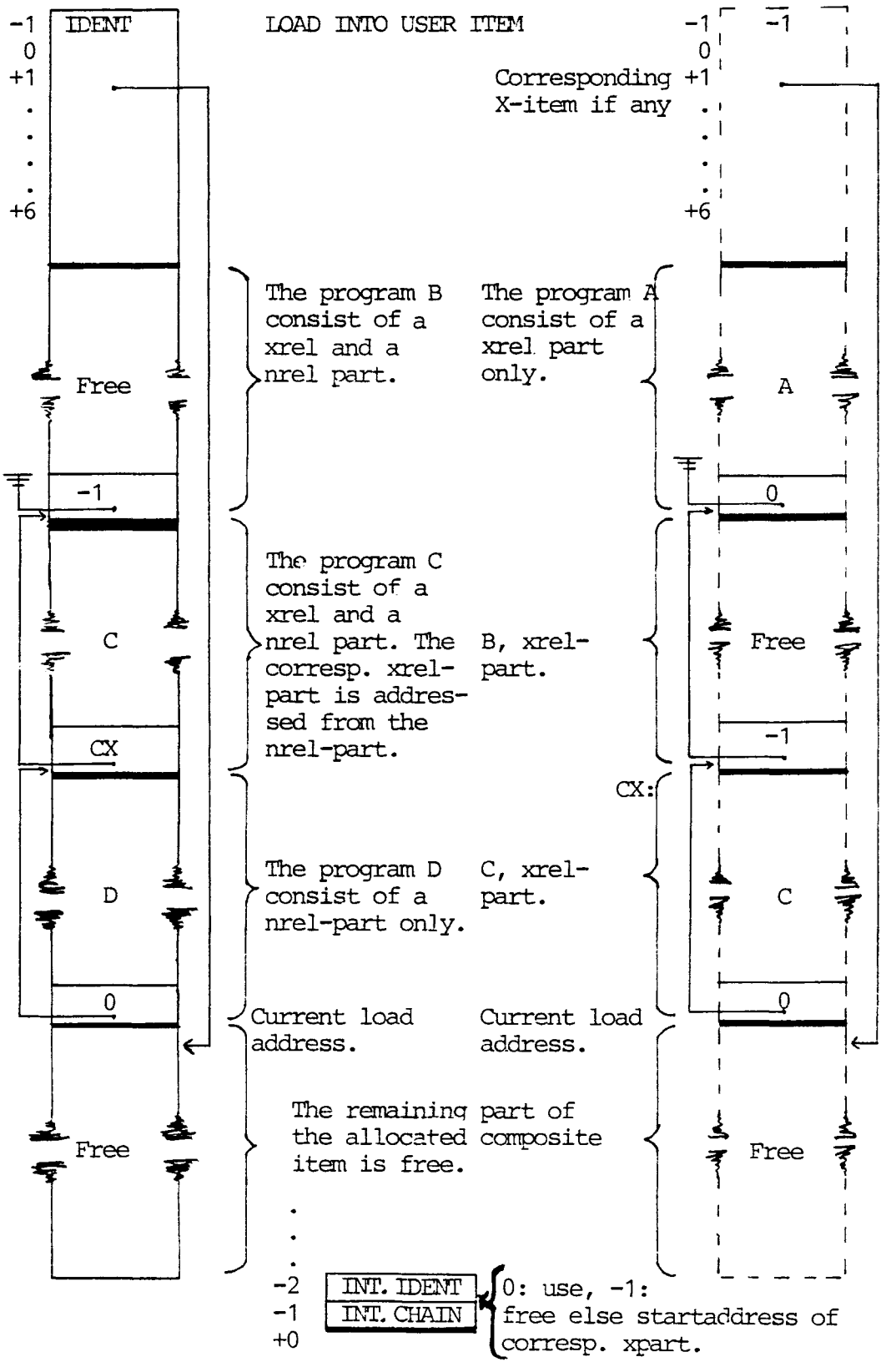


Ident = 0: N-item allocated only.

Ident = -1: X-item allocated only.

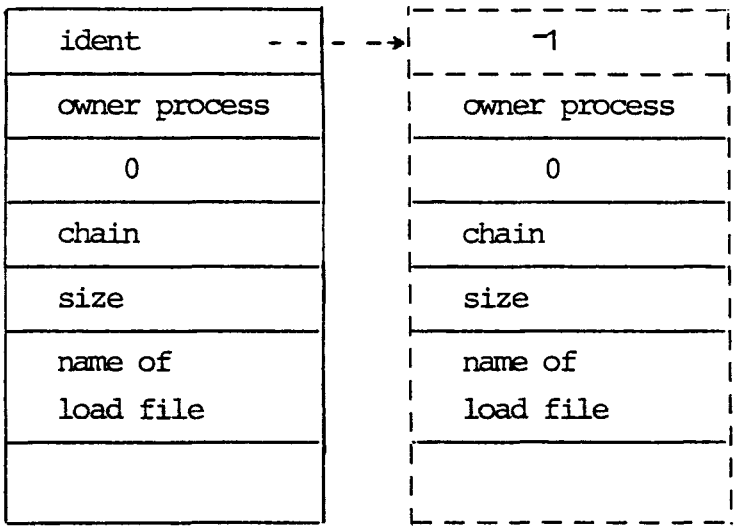
Ident <> 0,-1: A N-item and an X-item are allocated and the corresponding X-item is pointed out from the N-item. Both items are present in the core item chain and the corresponding X-item will always be last in chain.

In the case one or more processes/programs are loaded into one core item, the different parts of the item are pointed out using an internal chain. The actual program/process parts are kept together using an internal ident.



2.1.4 The Utility Core Item.

The item is allocated because of a single load



Ident = 0: N-item allocated only.

Ident = -1: X-item allocated only.

Ident <> 0, -1: A N-item and an X-item are allocated and the corresponding X-item is pointed out from the N-item.

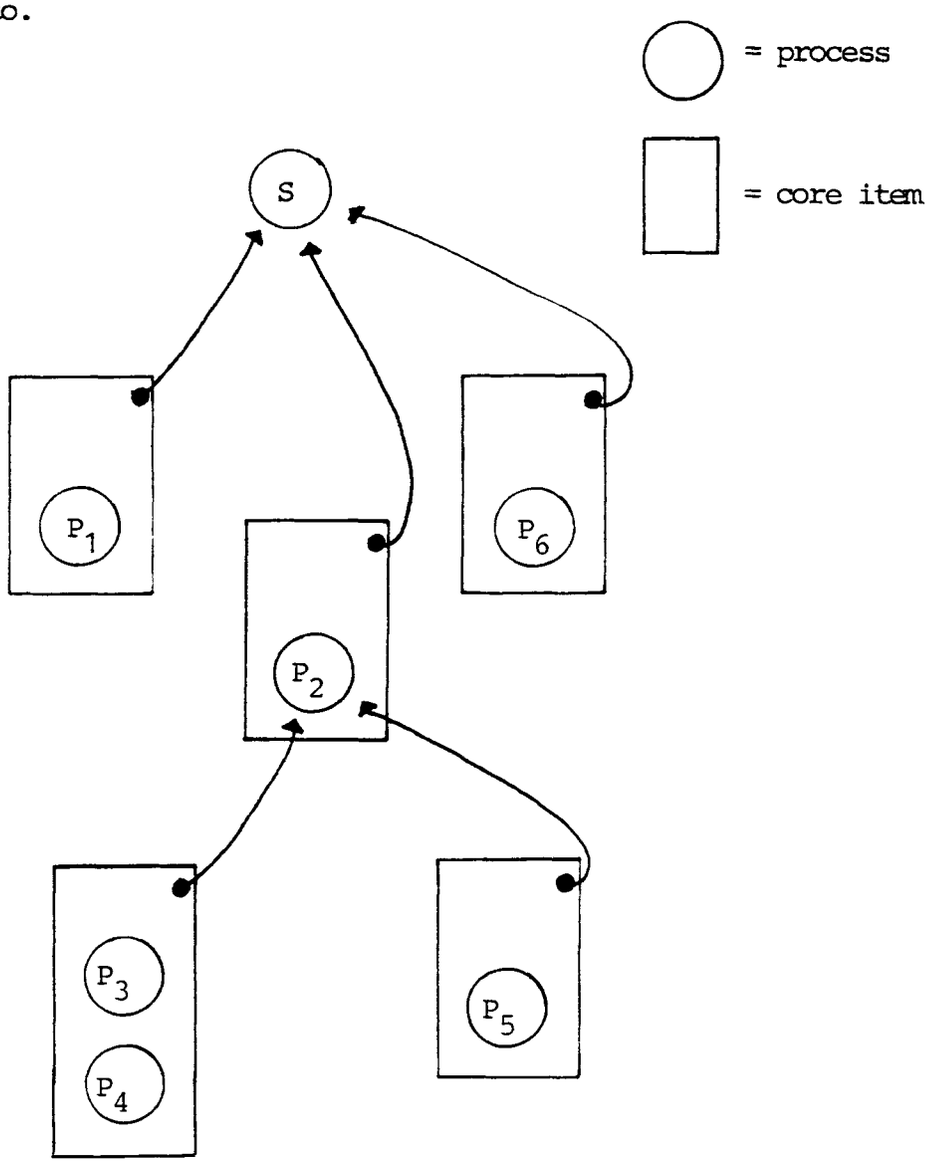
Both items are present in the core item chain and the corresponding X-item will always be last in chain.

2.2 Process Hierachy. 2.2

The above mentioned two relations between core items and processes introduce a relation between processes:

p_1 parent to p_2 = p_2 child of p_1: } there exists a core item C so that p_2 in C owned by p_1

All processes except for the processes in the basic system are children of other processes and all these processes are organized in a structure with respect to the relation parent to.



3. PROGRAMS AND PROCESSES.

3.1 Program Descriptor Word.

The following bits of PSPEC [1] are used by the system:

B5: parameter bit: if this bit is set, parameters are placed immediately after the highest N relocation occupied by the code. If the program contains a process descriptor the address of the parameter are placed in ac 1 when the process is started.

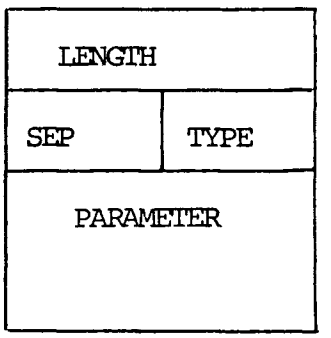
B6: paged program bit: the program is paged see ref [3]

B7: reservation bit: the process is a driver process. Processes using this program are breaked with cause = 8 if a reserver of the process is killed.

3.2 Program Parameters. 3.2

Program parameters are placed immediately after the loaded normal relocatable code, that is the last used address of the allocated N-item. If no N-item is allocated the parameters are placed immediately after the last used X-item location.

Program parameters consist of a sequence of items, each of the form:



An item always starts at an even byte address. The length of an item is the number of bytes in the item, odd or even. The separator is a byte containing the ASCII value of the separator preceeding the item or a zero if the item was preceeded by no separator. The type is an integer denoting the type of the item.

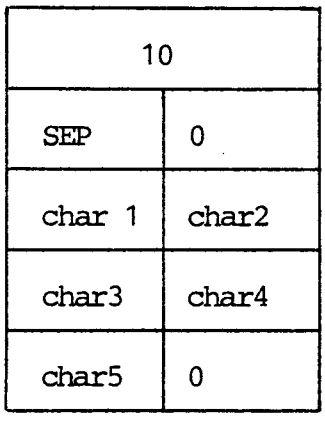
3.2.1 Separators. 3.2.1

The following values may appear in the separator field:

0:	blank, no separator.
44:	,
46:	.
47:	/
58:	:
61:	=

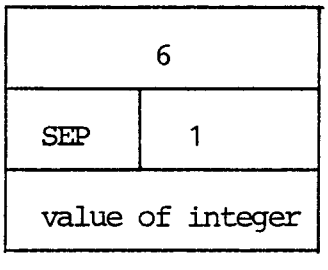
3.2.2 Item Types.

Type 0, names:

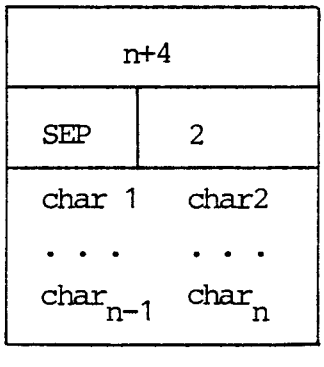


In a name item up to five characters of the name are placed, unused positions being zeroised. Only the first five characters of a name are significant.

Type 1, integer:

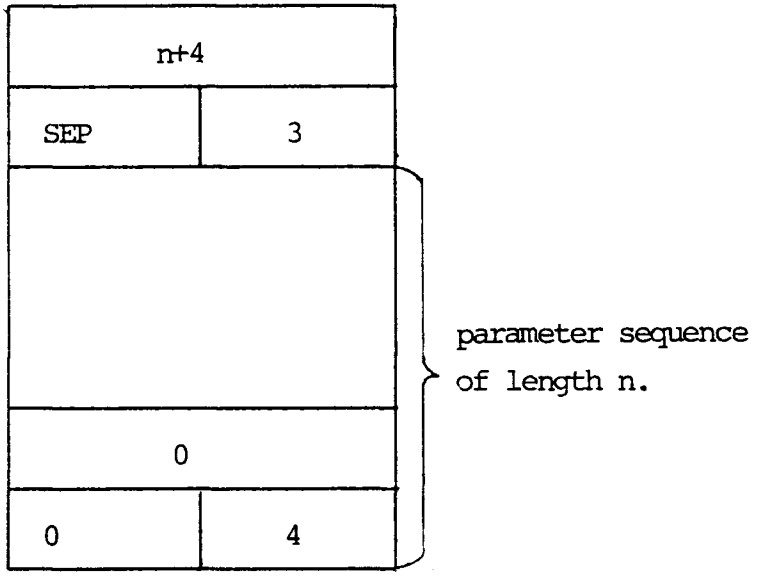


Type 2, texts:



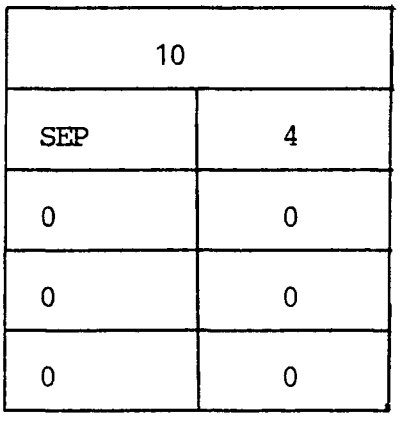
If n odd the slack byte is set to zero.

Type 3, composite item:

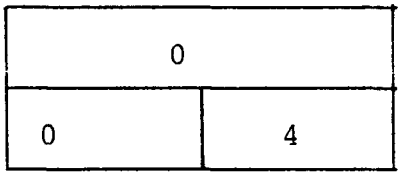


The composite item contains all parameters enclosed in parentheses. The sub sequence starts in adr (item)+2.

Type 4, dummy item:

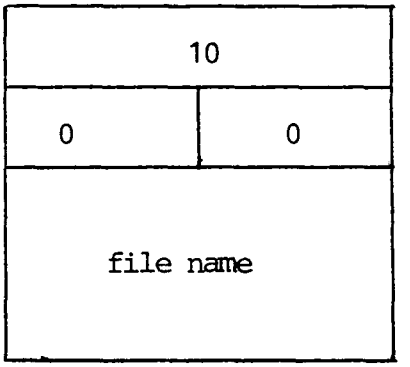


Type 5, end item:



3.2.3 Scanning the Parameters.

The first item contains the name of the filename from where the program was loaded:



The next item is found using the following algorithm:

```
;  ac2 = adr (item)
FNP:                    ; FETCH NEXT PARAMETER:
   LDA    0    0,2 ;       length:=item.length;
   INCZR  0,0      ;       size:=(length+1) /2;
   ADD    0,2      ;       adr:=adr+size;
;  ac2 = adr(next item)
```

Note that the end of the list is found when the length equals zero. Also note that this algorithm will never pass beyond the end item, but will continuously give you the end item.

3.2.4 Parameter Example. 3.2.4

COMMAND: PIP 987/'AB'

PARAMETERS:

10	
0	00
80	73
80	0
0	0
6	
0	1
987	
6	
47	2
65	66
0	
0	4

4. COMMUNICATION WITH S.

4.1 Event Classification.

After system initialization the process S goes into its idle state where it is waiting, ready to execute an S-function. As an event arrives, S classify it as being one of the following types of events.

1) Console command

An answer arrives from the teletype indicating that a human operator wants S to perform an S-function.

2) Internal command

An output message from a process in the system arrives indicating that a process wants S to perform a number of S-functions.

3) Internal request

A control message from a process in the system arrives indicating that a process wants S to perform a final operation on the process itself.

4) Get message

An input message from a process in the system arrives indicating that a process in the system wants S to deliver a message from the message file.

When executing console commands, the operating process is defined as S itself. When executing internal commands the operating process is the sender of the message. Internal requests are only executed if the sender of the message is a child of S, and the operating process is then defined as S

itself.

Generally S accepts to operate only on core items owned by the operating process. So the only processes S would kill is the children of the operating process. Some special functions, however, violate these rules. In [2] all functions are described. In appendix B a survey of commands are listed.

4.2 Error Messages. 4.2

Error messages consist of 3 components:

1. An error cause
2. Possibly a name
3. Possibly a number

When printing error messages on the teletype the error message would look like:

***<text> [<name>] [<number in octal>]

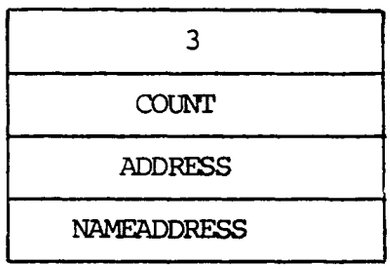
where the text explains the error cause.

When returning answers to internal messages the error cause is represented as a positive number, this number being the left part of the return value of Mess0. This value may be used as key accessing the error message file for a transfer of the actual error text.

4.3 Event Formats.

4.3.1 Internal Command.

Message:



The count is the number of characters to be interpreted by S.. The address is the byte address of the first character in the message. The name address is a byte address printing to a 6 byte core storage area or zero if no area present.

If <command string> denotes the contents of the bytes in locations address, address+1, ..., address+count-1, the command is interpreted as if the following commands were keyed in from the teletype:

BEGIN <nl><command string><nl>END<nl><end medium>

The scan terminates when the first END is met. Note that if the bytecount is accurate you need not put an END as the last command in the command string.

Answer:

RESULT * 256
COUNT
NUMBER
NAMEADDRESS

The result is zero if the commands were executed successfully otherwise it contains a positive error code, corresponding to an error text, refer to app. D. The count is equal to the count of the message. The number is the number part in case the error message equivalent to error code 22 is returned else zero, refer to app. E.

If the name field exists and contains a non full name, that name is the name part of the error message. Be careful not to send a message to S with an undefined value of mess 3.

4.3.2 Internal request. 4.3.2

Message:

1B8
COUNT
ADDRESS
NAMEADDRESS

The count and address acts as for internal commands. The command string is interpreted as if the following commands were keyed in from the teletype:

BEGIN <nl>KILL<sender><nl>
<command string><nl>END<nl><end medium>

Answer:

RESULT * 256
COUNT
NUMBER
NAMEADDRESS

The answer is given immediately and the process will be removed as fast as possible.

4.3.3 Get Message.

Message:

5
COUNT>32
ADDRESS
MESSAGENUMBER

This message asks for a transfer of the text of the system message with the number given in mess3. All system messages mentioned in ref [3] could be called for.

Answer:

RESULT*256
COUNT
NUMBER
0

The count is the number of characters in the text including a terminating zero byte. If count of message is too small, only a part of the text is delivered.

APPENDIX A - REFERENCES

[1] Mus System Introduction and Programmer's Guide.

[2] DOMUS User's Guide Part I

[3] DOMUS User's Guide Part II

[4] RC3600 Paging System, System Programmer's Guide.

[5] RC3600 CATALOG SYSTEM, System Programmer's Guide.

APPENDIX B - SYNTAX OF S-COMMANDS.

The following metalinguistic symbols are used:

Sequences of characters enclosed in < and > represent metalinguistic variables whose values are sequences of symbols. The mark ::= means 'may be composed of' and the mark | means 'or'. The production (rule): <sign> ::= + | - means that any occurence of the variable <sign> may be replaced by a + or a -. The braces < and > signifies that the contents should be regarded as a single metalinguistic variable. The superscription * means zero or more occurrencies of the preceeding variable, whereas the superscription + means one or more occurrencies. The brackets [and] indicates an optional string.

<command>	::=	<nl> * <name>{ [<sep>] <item>}* <nl>
<item>	::=	<name> | <number> | <text>|
		<dummy item> | <composite item>
<composite item>	::=	(<item>{ [<sep>] <item>} *)
<name>	::=	<letter> { <letter> | <digit>} *
<number>	::=	[<sign>] {<digit>$^{+}$'}* <digit>$^{+}$
<text>	::=	'<any character except'>'
<dummy item>	::=	*
<sep>	::=	. | / | : | = |,
<sign>	::=	+ | -
<letter>	::=	A |B| C | D| E| F| G| H| I| J| K| L| M| N|
		O |P| Q | R| S| T| U| V| W| X| Y| Z| Æ| Ø| Å|
		ß |
		a |b| c | d| e| f| g| h| i| j| k| l| m| n|
		o |p| q | r| s| t| u| v| w| x| y| z| æ| ø| å
<digit>	::=	0 |1| 2 | 3| 4| 5| 6| 7| 8| 9|
<nl>	::=	ASCII characters LF,VT,FF, or CR

The ASCII characters: SP and HT and the sequence ! <any characters except !>! are blind outside texts, except for being terminators of names and numbers.

APPENDIX C - SURVEY OF S-COMMANDS.

BEGIN	Read a sequnce of command lines from the teletype. Terminate at an END command.
BOOT <filename>	Bootstrap a stand-alone program.
BREAK <process>	Break the selected process.
CLEAN <process>	Stop and clean the selected process.
CLEAR [<coreitem>]	Kill all processes in the coreitem if specified else kill all processes in all utility coreitems.
CONNECT <subcaname>	Select the specified subcatalog at current subcatalog.
DRIVE <driveno>	Select the specified drive as current drive.
END	Terminate a sequence of commands and execute these commands.
FREE <coreitem>	Free the specified coreitem.
GET <coreitem> [<size> | <size> <size>]	Get the specified coreitem.
INIT <driveno>	Initialize the catalog on the disc drive.
INT <file>	Read a sequence of command lines from the specified file. Terminate at an END command.
KILL <process>	Kill the specified process.
LIST [/CORE | /PROGRAM] <name> *	List all or selected processes, programs or coreitems.
LOAD [/<coreitem> [/<size> | /<size> /<size>]] {{<file> | (<file> <params>)} [/<procname>]} +	Load a coreitem from the specified file(s).
RELEASE	Release current subcatalog and select the main catalog on current drive.

START <process>	Start the specified process.
STOP <process>	Stop the specified process.
<filename><params>	Load a utility coreitem from the specified file.
WAIT <Timer>	WAIT the specified number of secs.

APPENDIX D - SURVEY OF ERROR MESSAGES AND EQUIVALENT ERROR NUMBERS.

1	*** SYNTAX
2	*** TOO MANY PARENTHESES
3	*** PARAM
4	*** END MEDIUM, FILE <filename>
5	*** TOO MANY COMMANDS
6	*** STATUS, FILE <filename><status>
7	*** UNKNOWN, FILE <filename>
8	*** RESERVATION, FILE <filename>
9	*** COREITEM EXISTS, ITEM <itemname>
10	*** SIZE
11	*** COREITEM DOES NOT EXIST, ITEM <itemname>
12	*** COREITEM NOT CLEARED, ITEM <item>
13	*** ENTRY NOT A FILE, ENTRY <catalog entry>
14	*** STATUS, DEVICE <device name><status>
15	*** NOT ALLOWED
16	*** NO SPACE FOR PAGES, FILE <filename>
17	*** ILLEGAL PROGRAM, FILE <filename>
18	*** SIZE ERROR, FILE <filename>
19	*** CHECKSUM ERROR, FILE <filename>
20	*** VIRTUAL ADDRESS ERROR, FILE <filename>
21	*** PROCESS DOES NOT EXIST, PROCESS <process>
22	*** SYSTEM ERROR <number>
23	*** PROCESS EXISTS, PROCESS <process>
24	*** UNKNOWN, SUBCATALOG <filename>

APPENDIX E, THE 'SYSTEM ERROR' - MESSAGE.

The 'System error' - message is given the error no. 22 which is equivalent to the following message:
***SYSTEM ERROR <number>,
where the number refers to the following list:

Bootstrap errors

The error occurs during the bootstrap of the DOMUS system

1	Operator device malfunction
2	Master drive undefined
3	Master device malfunction
4	File management system malfunction
5	Paging file error
6	System configuration error
7	Error message file error
10-16	System malfunction

Runtime errors

21	Internal request error. An error occurs after the sending process is removed. Non fatal error.
22	DOMUS stack overflow. An implementation restriction has been violated. Fatal error, the system may fail to operate properly.
24	Error message file error. Fatal error, the system may fail to operate properly.
25	Core storage structure destroyed. Fatal error, the system may fail to operate properly.

APPENDIX F SURVEY OF CORE ADMINISTRATION

Fig. 1, coreitem and program relationship:

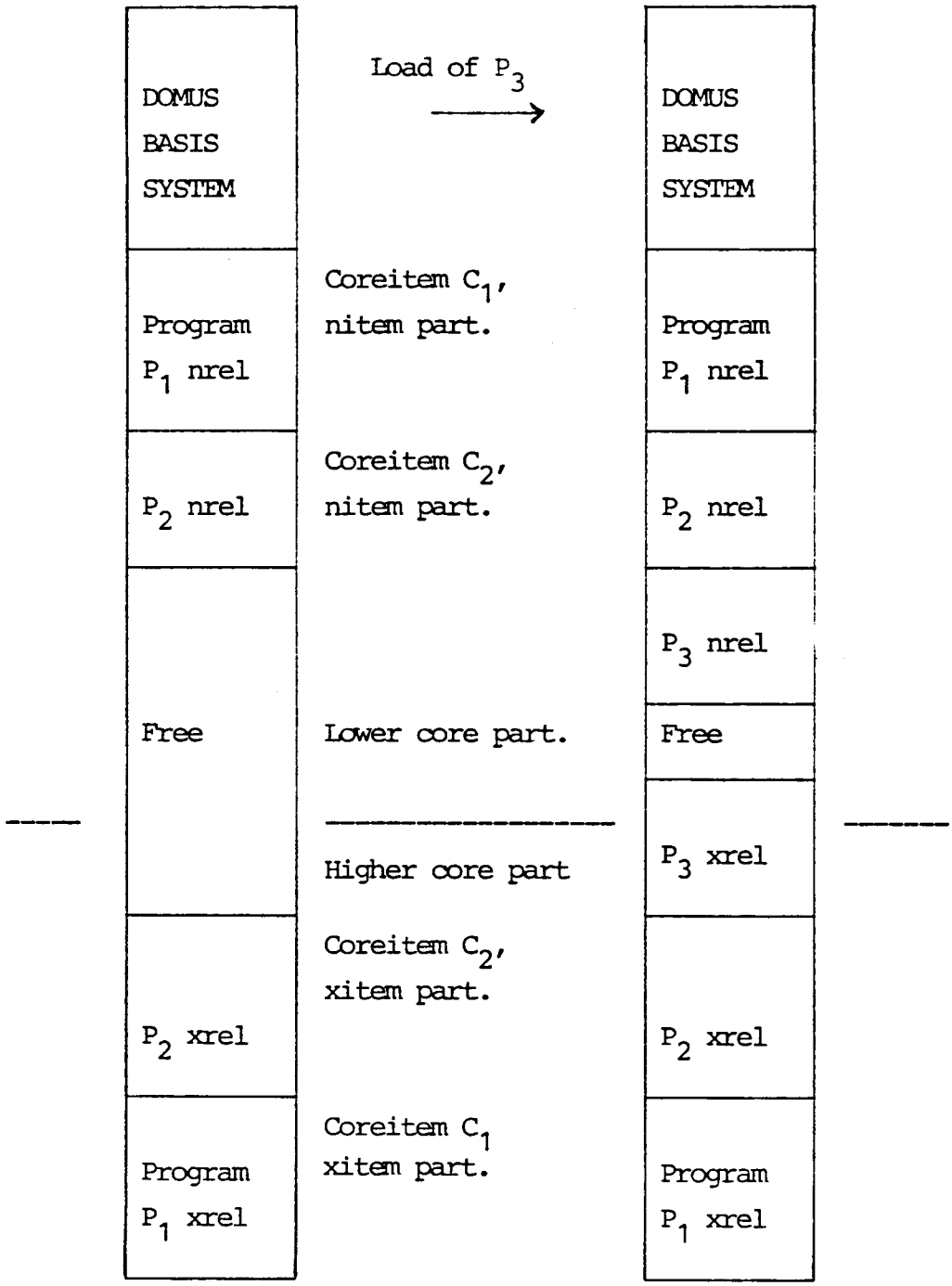


Fig. 2, storage allocation scheme: