
RCSL No: 31-D662

Edition: February 1982

Author: Kirsten Kjøller Hansen
Niels Møller Jørgensen
Lars Otto Kjær Nielsen
Edith Rosenberg

Title:

Swopping Online System (SOS)

User's Guide/Reference Manual/Operating Guide/Installation Guide

Keywords:

RC8000, RC4000, operating system, interactive program execution, terminal access.

Abstract:

This manual describes the operating system SOS. The manual contains information relevant to users, programmers, operators and system staff.

(86 printed pages)

Copyright © 1982, A/S Regnecentralen af 1979
RC Computer A/S
Printed by A/S Regnecentralen af 1979, Copenhagen

Users of this manual are cautioned that the specifications contained herein are subject to change by RC at any time without prior notice. RC is not responsible for typographical or arithmetic errors which may appear in this manual and shall not be responsible for any damages caused by reliance on any of the materials presented.

FOREWORD

First edition: RCSL No 31-D482.

Second edition: RCSL No 31-D512.

This manual describes the operating system SOS, as it appears in release 2.0, October 1978. The former version was a preliminary version intended for multiterminal jobs. The present version includes interactive multiterminal jobs (using TEM), interactive single terminal jobs (with or without using TEM) and batch jobs. The present version supports in a primitive form access to devices like tape stations and flexible discs.

This manual is an exhaustive description of SOS and it includes subjects of interest to users, programmers, operators and installation staff.

Niels Møller Jørgensen

A/S REGNECENTRALEN, September 1978

Third edition: RCSL No 31-D662.

The changes are indicated by correction lines in the left margin and concern mainly the description of the new facility for typing invisible password.

Edith Rosenberg

A/S REGNECENTRALEN af 1979, October 1981

TABLE OF CONTENTS

PAGE

1. INTRODUCTION	1
2. USER'S Guide	4
2.1 SOS Illustrated by Means of Examples	4
2.2 SOS User Commands	13
2.2.1 Job Creation Commands	14
2.2.2 Multiterminal Commands	15
2.2.3 Job Intervention Commands	17
2.2.4 Device Handling Commands	18
2.2.5 Password	19
2.3 Terminal Access	20
2.4 Resource Allocated to SOS Jobs	21
3. REFERENCE INFORMATION	22
3.1 Job Scheduling	22
3.2 Terminal Access	25
3.3 SOS - Job Intercommunication	26
3.3.1 Primary Input and Output	27
3.3.2 Transparent Communication to TEM	28
3.3.3 Parent Messages	30
4. OPERATING GUIDE	33
4.1 Start-Up	33
4.2 Operator Intervention	34
4.2.1 System Intervention	34
4.2.2 Job Intervention	35
4.2.3 Device Support	36
4.3 Close Down	37
4.4 How to Handle System Failure	37
5. INSTALLATION NOTES	40
5.1 System Distribution	40
5.2 System Trimming	41
5.3 User Catalog	48
5.3.1 Data per Job (Process)	49

<u>TABLE OF CONTENTS (continued)</u>	<u>PAGE</u>
5.3.2 Creation and Change of the User Catalog	50
5.3.2.1 Creation	51
5.3.2.2 Change	53
5.3.2.3 Listing	55
5.3.3 Resources Needed for Creating a Catalog	55
5.3.4 Error Messages	56
5.4 Test Facilities	59

APPENDICES:

A. REFERENCES	63
B. SOS COMMANDS	64
C. MESSAGES FROM SOS	65
D. AN EXAMPLE OF A MULTITERMINAL PROGRAM	71
E. EXAMPLE OF A USER CATALOG	72
F. THE AUXILIARY FILE "sostrim"	73
G. THREE VERSIONS OF A MASTER MIND PROGRAM	75
G.1 Simple Single Terminal Version	75
G.2 Single Terminal Version with Input Checking	76
G.3 Multiterminal Version with Input Checking	77
H. INDEX	78
H.1 Survey of Examples	78

1. INTRODUCTION

1.

The main purpose of an operating system is to serve the users by supervising the execution of their programs. In a multiprogramming environment with a number of users running their programs simultaneously the main tasks include:

- 1) facilitate resource sharing
- 2) control and facilitate the access to common equipment
- 3) prevent unauthorized access to private information

In the case of RC8000 the supervisory functions are performed by the monitor (ref. [1]). The resources of the computer may be partitioned into so-called processes. A process is a set of resources used for program execution. In each process programs may be executed one at a time. A process may play the role of an operating system by creating and controlling child processes according to some strategy.

SOS is an operating system process and each user logged in is served by a child process, created by SOS.

In many computer systems the primary store is a key resource. To obtain a good machine utilization it is important that key resources are released, when they are not actively used.

At interactive program execution there will be long periods during which the process waits for the terminal operator to type some input. In case no resources are released during these periods, the machine utilization will become prohibitively poor.

The main purpose of SOS is to make a number of programs (primarily interactive programs) share the same part of primary store. This is done by supervising the program execution and by swapping programs (writing the primary store to backing store during passive periods).

SOS is mainly intended for supporting interactive program execution. However, SOS includes facilities for executing batch

jobs, too, and the resource scheduling is based on the following elements:

A series of programs executed in the same process one after the other is called a "job".

- a) Any job running under SOS is in one of the following states:
 - 1) running
 - 2) suspended (ready to run, but the primary store is occupied by another job)
 - 3) waiting (the job is not ready to run - e.g. waiting for terminal input)
- b) Execution time is allocated in slices (of a few seconds). In case a job does not suspend itself within a time slice it will be suspended by SOS.
- c) Highly interactive jobs will be favoured by a high priority, while more CPU-bound jobs will get decreased priority. The time slices are allocated according to the priorities of all jobs which are ready to run.
- d) Batch (or background) jobs are activated only during periods where no interactive jobs are ready to run, and a running batch job is stopped immediately when some interactive job becomes ready for activation (for instance when input arrives from a terminal).

In RC8000 any program (utility or user defined program) is executed in a so-called "process", no matter the operating system controlling the execution. Thus the access to the basic computing facilities (CPU, primary store and backing store) is not affected by the actual operating system, and all utility programs like compilers, editor, catalog handling programs etc. may be used under SOS exactly as under S or BOSS.

However, SOS does not allow full and direct access to slow devices like printers, paper tape readers etc. This is so because

the resources occupied by programs using slow devices are often poorly utilized - it is extremely inconvenient if interactive programs are blocked by programs using slow devices.

The use of slow devices is assumed to be undertaken by service modules like the printing module (PRIMO) or the file router.

2. USER'S GUIDE

2.

The present chapter illustrates by means of examples some of the facilities offered by SOS. All facilities of SOS are described in detail in later sections. According to the intentions of SOS the examples mainly concern interactive program execution.

2.1 SOS Illustrated by Means of Examples

2.1

Job creation.

SOS is always accessed from a terminal and a session is initiated by a "job creation". At job creation SOS consults the user catalog to check that the user is allowed to use the system and to know how much resources shall be allocated to him. The job creation is initiated by pushing the attention key. The user is then asked for the name of the system, he is going to use. In case of SOS the user will be answered by a ">" and then he may type his command. (In the examples all output written on the terminal is printed with capital letters while terminal input is printed with small letters. @ denotes a push on the attention key).

@

ATT sos

> go rc pass nn

02.06 SOS: RC ENROLLED

Example 1: Job creation.

The command (go) creates a set of resources (a process) with the name "rc" and initiates this process by loading the file processor FP (see ref. [3]). In this example a password ("nn") is demanded to create the job "rc".

When a user has started a session by creating a job as explained in the former, his next command(s) will be interpreted by FP and he will be able to run the utility programs interactively.

claim

AREA 24 BUF 25 SIZE 60416 FIRST CORE 65268

DISC: 21 SEGM/Slice

TEMP 504 SEGM 18 ENTR

LOGIN 315 SEGM 5 ENTR

PERM 315 SEGM 5 ENTR

Example 2: Executing utility program.

Simple program development.

The present part of the chapter primarily deals with interactive program development (typing and debugging), and readers who are not familiar with or not interested in program development, may skip this section and continue at "execution of interactive programs".

Assume that some user wants to exercise his mind by playing the "master mind" game. Then he needs someone to set up random combinations and to answer his guesses. He realizes that in case he could write a program enabling the computer to do the job, he would have the fastest, most reliable and most patient player ever seen.

After some thinking the user has created a program in his mind (and on a piece of paper) and he wants to debug it. First of all he must type in the program text. This is done by using the utility program "edit".

```

mastertext=edit
EDIT BEGIN.
i/
begin
    integer array solution,guess(1:4);
    integer i,j,x,digitok,digitincluded;

setcombination:
    for i:=1 step 1 until 4 do
    begin
        random(x);
        solution(i):=x mod 10;
        for j:=1 step 1 until i do
            if solution(i) = solution(j) and i < j then i:=i-1;
        end;
        write(out,<:master mind program ready<10>:>);

next:
    setposition(out,0,0);
    write(out,<:          :>);
    for i:=1 step 1 until 4 do read(in;guess(i));
    digitok:=digitincluded:=0;
    for i:=1 step 1 until 4 do
    begin
        for j:=1 step 1 until 4 do
            if guess(i) = solution(j) then
            begin
                if i=j then digitok:=digitok+1
                else digitincluded:=digitincluded+1;
            end;
        write(out,<<dd>,guess(i));
    end;
    write(out,<: => :>,false add 43,digitok,
        false add 32,4-digitok,
        false add 45,digitincluded,
        false add 10,1);
    if digitok <4 then goto next else
    begin
        write (out,<:you got it !!<10>:>);
        goto setcombination;
    end;
    /,
    f
    EDIT END.

```

Example 3: Creating a text file.

Now the program text has been created and stored in the file "mastertext". To save the text for later use it may be permanented by means of the utility program "scope":

```
scope user mastertext
lookup mastertext
MASTERTXT =SET 2 DISC D.780930.0207 0 0 0 0 ; USER
; 1886 135 3 810 819
```

Example 4: Executing utility programs.

After this the program may be translated by calling the ALGOL compiler.

```
masterprog = algol mastertext

MASTERTXT D.780930.0207
1:BEGIN

37:END;
6. LINE 10 . 3 UNDECLARED
LINE 16 . 6 TERMINATION
LINE 16 . 8 DELIMITER
ALGOL END 22
```

Example 5: Program translation.

The errors detected by the compiler may be corrected by means of the editor, like this:

```
newmaster = edit mastertext
EDIT BEGIN.
1./solytion/, r/solyt/solut/
IF SOLUTION(I) = SOLUTION(J) AND I <> J THEN I:=I-1;
1./read(in;/
FOR I:=1 STEP 1 UNTIL 4 DO READ(IN,GUESS(I));
r/in;/in,/
FOR I:=1 STEP 1 UNTIL 4 DO READ(IN,GUESS(I));
f
EDIT END.
mastertext=move newmaster
masterprog = algol mastertext

MASTERTXT D.780930.0217
1:BEGIN

37:END;
ALGOL END 25
```

Example 6: Text editing.

The corrected version of the program is translated and in case the translation was successful the next step will be to run the program.

Execution of interactive programs.

All programs (utility or user defined) are activated by the file processor which reads the job control commands (program calls) loads the actual programs and hands over eventual parameters. The master mind program just created uses no parameters and it may be executed like this:

```

masterprog
MASTER MIND PROGRAM READY
0 1 2 3
4 5 6 7      0 1 2 3 =>  —
1 0 7 8      4 5 6 7 =>  -
7 9 1 0      1 0 7 8 =>  —
7 8 0 2      7 9 1 0 => + -
6 2 8 0      7 8 0 2 =>  —
3 7 8 0      6 2 8 0 => ++
              3 7 8 0 => ++++
YOU GOT IT !!
MASTER MIND PROGRAM READY

```

Example 7: Execution of interactive program.

The master mind program is by its nature an interactive program as it would not be possible to prepare the input (guesses) in advance - each new input will depend on all answers written by the program.

The program developed will never terminate. When the user reaches the solution the program will generate a new one and try again. The only way to escape is to make a job intervention. To remove the job the user may proceed like this:

```
@
ATT sos

>kill
02.23 SOS: READY
02.23 SOS: RC REMOVED AFTER USER KILL
```

Example 8: Job intervention.

Job termination.

However, most programs will terminate, and the usual way to close a terminal session is by calling the utility program "finis".

This program tells SOS that the job has finished and the resources allocated may be released.

```
finis
02.24 SOS: RC REMOVED AFTER FINIS
```

Example 9: Job termination.

The master mind program may be changed in different ways. In appendix G is shown two new versions. One of them is a single terminal version performing input check. The other one is a multi-terminal version based on the context facility of ALGOL and accessing the terminals via TEM.

Accessing a Multiterminal Program.

A Multiterminal version of the mastermind program (as shown in appendix G) may be accessed from a terminal by using the multi-

terminal "login" and "logout" facilities of SOS. The session may look like this (the program is executed in the job "team"):

```
@
ATT sos

>in team usera pass al
02.52 SOS: TERMINAL CONNECTED

FROM TEM
MASTER MIND PROGRAM READY
0 1 2 3
4 5 6 7
8 9 1 5
9 8 2 5
9 3 8 5
YOU GOT IT !!
MASTER MIND PROGRAM READY
```

	0 1 2 3 =>	-
	4 5 6 7 =>	-
	8 9 1 5 => +	-
	9 8 2 5 => ++	-
	9 3 8 5 => +++	

Example 10: Running a multiterminal program.

Automatic Program Activation.

In the examples shown so far the user has controlled his activities by calling the programs one by one. In case a user from time to time runs exactly the same set of programs exactly the same way, he may get his programs activated automatically. This may be done by stating in the user catalog a set of commands that will be interpreted at login.

Some service functions may be implemented by executing automatically activated programs. A program, listing the jobs currently enrolled, might be run this way:

```
@
ATT sos

>run display

FROM TEM
02.25 SOS: DISPLAY ENROLLED

DISPLAY AF SOS DEN 78 09 30 KL. 2.25

          CLAIM:
NAVN      START  PORT  CPU   SIZE  BUF AREA
RCSAVE    02.24  02.24  0.0   75264  3    6
TEAM      02.25  02.25  0.0   75264  3    6
DISPLAY   02.25  02.25  0.1   12800  3    5

END      21
02.25 SOS: DISPLAY REMOVED AFTER FINIS
```

Example 11: Automatic program activation.

Running batch jobs

A number of activities are not suited for interactive execution. Often these activities are running for a long time without any communication with the user. Programs generating safety copies, translation of large systems or large database reorganizations are all examples of typical batch programs. Under SOS, batch programs may be executed, too. However, batch programs will never be activated when interactive programs are ready for running.

A user who wants to "save" his files on a magnetic tape may do like this:

```
@
ATT sos

>batch rcsave
02.27 SOS: RCSAVE ENROLLED
save mtlk0001.1.label.private tsos ttem

NO DUMPLABEL ON FILE 1
WRITTEN: DUMP MTLK0001 001 VERS. 300978.02 S=1 PRIVATE

TSOS          144 PROJECT.DISC      D.780929.2137
TTEM          67 PROJECT.DISC      D.780926.2015
  2 ENTR.,    211 SEGM.

WRITTEN: DUMP MTLK0001 002 EMPTY 300978.02 PRIVATE

END 75
finis
02.29 SOS: RCSAVE REMOVED AFTER FINIS
```

Example 12: Running a batch job from a terminal.

In case many interactive users are active in the system or in case the user saves many large files he will have to spend rather a long time between the "save-command" and the "finis command".

Instead he might prepare the whole job by creating a jobfile with the same contents:

```
savejob=edit
EDIT BEGIN
i/
save mtlk0001.1.label.private tsos ttem
finis
/,f
EDIT END.
scope project savejob
```

Example 13: Creating a job file.

The whole job may now be executed using this jobfile. By doing so, the user does not have to stay at the terminal during the run.

```

@
ATT sos

>batch rc jobfile savejob pass nn
02.32 SOS: RC ENROLLED

READ   : DUMP  MTLK0001   001  VERS. 300978.02  S=1 PRIVATE

WRITTEN: DUMP  MTLK0001   001  VERS. 300978.02  S=1 PRIVATE

TSOS           147      USER.DISC           D.780930.0137
TTEM           67      PROJEKT.DISC          D.780926.2015
  2 ENTR.,      214 SEGM.

WRITTEN: DUMP  MTLK0001   002  EMPTY 300978.02      PRIVATE

END      75
02.34 SOS: RC REMOVED AFTER FINIS

```

Example 14: Running a job using a job file.

Often it would be more convenient if output from batch jobs were printed on a line printer. This may be done by using the service module PRIMO (see ref. [5]). The output is currently written into a backing storage file and at job termination, PRIMO is asked to print the file.

```

savejob=edit
EDIT BEGIN
i/
o outfile
save mtlk0001.1.label.private tsos ttem
o c
scope user outfile
filexfer outfile lp queue.paper.a4
finis
/,f
EDIT END

```

Example 15: Job file routing job output to printer.

In this section all SOS user commands are described, their syntax and their semantics. The commands are grouped according to their functions. In chapter 4 of this manual (the Operating Guide) the SOS commands used at operator intervention are described. In case

the system is run by the users themselves at least some of the users will have to know how to use these operator intervention commands.

2.2.1 Job Creation Commands

2.2.1

SOS has three different job creation commands: "go", "run" and "batch". The syntax for the job creation commands is:

$$\left\{ \begin{array}{l} \text{go} \\ \text{run} \\ \text{batch} \end{array} \right\} \langle \text{jobname} \rangle \left\{ \begin{array}{l} \text{jobfile} \langle \text{filename} \rangle \\ \end{array} \right\} \begin{array}{l} 1 \\ 0 \end{array}$$

$$\left\{ \begin{array}{l} \text{pass} \langle \text{password} \rangle \\ \end{array} \right\} \left\{ \begin{array}{l} \text{newpass} \langle \text{password} \rangle \\ \end{array} \right\} \begin{array}{l} 1 \\ 0 \end{array} \left\{ \begin{array}{l} 1 \\ 0 \end{array} \right\}$$

See examples 1, 11, 12 and 14.

Each of these commands will create a job (a process) with a set of resources as described in the user catalog. A job started by the "go" or the "run"-command will be scheduled as an interactive job while a "batch" job will be scheduled as a background job (i.e. a "batch" job may only be active during periods where no "go" or "run" jobs are ready for activation).

The difference between the "go" and the "run" commands concerns the handling of the terminal only. In case of a "run" command the terminal is connected via TEM (see ref. [4]), offering output spooling. In case of a "go" command, the terminal is accessed by SOS directly - and no spooling is offered. At "batch" jobs, the terminal is always accessed by SOS directly (as at "go").

The "jobfile" option enables the user to prepare his job by creating a (permanent) file containing all program calls etc. necessary for the execution of his job. The jobfile is said to be the primary input document of the job. At jobs not using the jobfile option, the terminal is the primary input document. In all cases the terminal is the primary output document of the job.

To avoid confusion, SOS will not accept that a given terminal is the primary input document of more than one job at a time. However, the same terminal may be the primary output document of several jobs.

In other words, several jobs may be started from the same terminal (and run simultaneously), but at most one of these jobs may be created without using the jobfile option.

In the description of a process (a job) in the user catalog a field defining a password is included. In case this password is nonempty, the job creation will not be accepted unless the command includes: `pass <password>`. This facility is included to check the authorization of the user and to grant privacy. An improved security may be obtained by redefining the password. This may be done by using the "newpass" option. By means of this option, the password may be changed at startup like this:

```
... pass hobo newpass hifi
```

(if the password of a process has become empty, it is not possible by the newpass option to create a password - thus "public" jobs using no password cannot be blocked by a user defining a new password).

2.2.2 Multiterminal Commands

2.2.2

Jobs running multiterminal programs may use TEM (see ref. [4]) directly (i.e. transparently via SOS). In any case the job must ask TEM to create a terminal group (pool) and then the terminals may be connected to the group (i.e. links are created to the terminals). The terminal connections may be established or removed by the program explicitly asking TEM to create or remove the links. Principally the same job may be performed by SOS using the multiterminal commands "in" and "out".

A user who wants to be serviced by a multiterminal job may get connected by using the "in" command:

$$\text{in } \langle \text{jobname} \rangle \langle \text{username} \rangle \left\{ \begin{array}{l} \text{pass } \langle \text{password} \rangle \\ \text{newpass } \langle \text{password} \rangle \end{array} \right\} \begin{array}{l} 1 \\ 0 \end{array} \left\{ \begin{array}{l} 1 \\ 0 \end{array} \right\}$$

See example 10.

SOS then consults the user catalog to check that the jobdescription according to $\langle \text{jobname} \rangle$ includes a user description with the name $\langle \text{username} \rangle$.

The password and newpass options may be used here exactly as at the job creation commands.

After having connected the terminal, SOS generates a special input line and hands it to the multiterminal job. This input line is generated to tell the job that a new terminal has "logged in".

When the terminal user wants to leave the job, he may use the "out" command:

out

This command causes SOS to generate a special input line (as at "in") to tell the multiterminal job that the terminal "logs out".

The exact contents of the input lines generated at "in" and "out" may be found in chapter 3 (the Reference Part).

SOS includes a set of job intervention commands enabling the user to control his job fully. The commands are:

$$\left\{ \begin{array}{l} \text{stop} \\ \text{start} \\ \text{break} \\ \text{kill} \end{array} \right\} \left\{ \begin{array}{l} \langle \text{jobname} \rangle \end{array} \right\} \begin{array}{l} 1 \\ 0 \end{array}$$

See example 8.

The $\langle \text{jobname} \rangle$ must be stated in case the actual job is created using the "jobfile" option. In other words: if the terminal is the primary input document of the job, the $\langle \text{jobname} \rangle$ is not necessary.

In any case the job intervened must be created from the terminal that is used for the job intervention.

The "stop" command is used to suspend a job during execution. SOS will suspend the job in such a way that it will be possible to resume the job execution later on (in case input/output to or via SOS is programmed in a nonstandard way, data may be lost because of the intervention, but programs using standard I/O should continue unaffectedly).

The "start" command is used to activate suspended jobs. The suspension may be caused by a "stop" command, a "pause" parent message (see chapter 3) or the like.

The "break" command is used to interrupt a (possibly defective) program. SOS restarts the program in its interrupt routine, allowing it to run for at most one time slice. (Standard programs will use this piece of time to make some cleaning like: empty buffers, write error message etc.) After this period SOS aborts the job.

However, SOS may be trimmed in such a way that the break command merely works as "stop load start" when the job is enrolled without a jobfile. So the user will not lose his temporary files if he breaks a program.

The "kill" command is used to abort jobs. The actual job will be stopped immediately and there will be no time for the job to report errors or the like.

It should be mentioned here that the job intervention commands described in this subsection also exist in a special version as "operator commands". (The operator version of the commands is described in chapter 4).

2.2.4 Device Handling Commands

2.2.4

As SOS is primarily intended for interactive processing, the use of devices (apart from backing store) must be limited to a minimum. However, batch jobs will never block interactive jobs so it is acceptable to allow batch jobs to use some kinds of devices.

The most obvious need for device access concern printing (on line printer) and generation of safety copies on magnetic tape or flexible disc. Printer handling is supposed to be performed by a service module like PRIMO (printing module). Generation of back up, however, may be done by batch jobs.

The use of magnetic tape, flexible disc and other devices with exchangeable documents is supported by a "call" command and an "include" command.

The "call" command looks like this:

```
call <deviceno> <documentname>
```

The function performed by the "call" command is to name a device (specified by the device number: <devno>). After having named the device, all jobs enrolled to SOS at present will be included as

users of the device. At magnetic tapes and flexible discs the device (station or drive) is usually given the name of the document currently mounted.

In case of direct access to devices with fixed names (paper tape punch for instance), the job will have to be included as a user of the device. This may be done by using the "include" command:

```
include <deviceno>
```

All jobs enrolled to SOS at the moment of the inclusion will get access to the device - but still the device may only be used (reserved) by one job at a time.

It should be mentioned here that direct access to slow devices like a paper tape punch may cause a poor resource utilization, and only batch jobs should use this facility. Direct access to slow input devices like paper tape readers will not do neither at batch nor at online jobs.

2.2.5 Password

2.2.5

Instead of typing the password as described in subsections 2.2.1 and 2.2.2, the password may be typed invisible in a separate line.

If the password information in the login command is omitted, and the password defined for the job in the user catalog is non-empty, then SOS will answer by writing on the terminal:

```
>password
```

Now the password can be typed without echo on the screen, but a possible typing error cannot be corrected by means of backspace or rubout. The answer is delayed until a timer interrupt has occurred (about 5 seconds).

In this case the password cannot be changed by a newpass-command.

So the syntax for job creation commands using invisible password is:

$$\left\{ \begin{array}{l} \text{go} \\ \text{run} \\ \text{batch} \end{array} \right\} \langle \text{jobname} \rangle \left\{ \begin{array}{l} \text{jobfile} \langle \text{filename} \rangle \\ \end{array} \right\} \begin{array}{l} 1 \\ 0 \end{array} \text{'nl'}$$

">password" <password>

where the text in " " is written by SOS.

The syntax for the "in" command becomes:

in <jobname> <username> 'nl'
">password" <password>

Note! When the terminal is the main console, the password will not be invisible, neither when the console is a screen.

2.3 Terminal Access

2.3

A program executed under SOS may perform terminal access in different ways depending on job creation and program behaviour.

Jobs created by the "run" command will access the creating terminal via SOS and TEM, using the spooling facilities of TEM. Jobs created by "go" or "batch" commands will access the creating terminal via SOS only.

Any job (created by "run", "go" or "batch" command) using the "jobfile" option will use the creating terminal for output only.

In any case the creating terminal is usually accessed by using the standard zones "in" and "out".

Any job may access one or more terminals via TEM by using the TEM facilities directly (see ref. [4]). However, all TEM operations

will be communicated via SOS, but they are handled almost transparently so that the user will normally feel no change. (The differences are described in chapter 3).

2.4 Resources Allocated to SOS Jobs

2.4

According to the main intentions of SOS, the jobs should not be slowed down by accessing slow devices (and thereby block the primary store for considerable amounts of time).

A reasonable rule for resource allocation may look like this:

- A: interactive jobs ("go" or "run" jobs) should never access peripherals slower than backing storage.
- B: batch jobs should never access peripherals slower than flexible discs or magnetic tape.
- C: access to slow peripherals like printers should always be done via a service module like "PRIMO".

As a consequence the resources described in the user catalog only concern:

- 1) message buffers
- 2) area processes
- 3) backing storage entries
- 4) Backing storage segments

The direct use of devices (apart from backing storage) is not supported by the job creation (or the user catalog) - the user himself (or the operator) must use the "call" or "include" commands for this purpose.

3. REFERENCE INFORMATION

3.

The present chapter contains exact descriptions of subjects and details that most users do not have to consider. However, it may be necessary to go into some details to understand the system behaviour fully.

3.1 Job Scheduling

3.1

SOS handles two kinds of jobs: interactive jobs and batch (or background) jobs. As a consequence the job scheduling is based on two main queues: a queue of interactive jobs and a queue of batch jobs. These two queues contain only jobs which are ready to run. Jobs waiting for external events like terminal input are queued up in a waiting queue.

Only one job may be active at a time and this job is found like this:

- If the interactive queue is not empty, then find the "best" job and activate it.
- If the interactive queue is empty and the batch queue is not empty, then take the first job in the batch queue and activate it.
- If both queues are empty, then wait for an event making some job ready for execution.

When started, a job will be allowed to run for a period that depends on the kind of job.

1. An interactive job will be stopped
 - 1) when it asks for input,
 - 2) when its terminal output exceeds the spooling capacity,
 - 3) when the time slice expires or
 - 4) when the job ends - whatever happens first.

2. A batch job will be stopped
 - 1) when it asks for input,
 - 2) when its terminal output exceeds the spooling capacity,
 - 3) when the job ends or
 - 4) when some event arrives that will make an interactive job ready to run - whatever happens first.

When a batch job is stopped because of an interactive job, it will remain the first job in the batch queue. When a batch job is created it is put back on the batch queue. Thus the strategy of the batch job scheduling is a pure first in first out strategy.

However, this strategy may be overwritten by users or operators using the job intervention commands "stop" and "start". "Stop" will move the job from the batch queue to the waiting queue. "Start" will move the job to the batch queue but in different ways depending on circumstances:

operator start moves the job to the head of the queue
user start puts the job back on the queue.

Interactive jobs are scheduled according to dynamically changing priorities. Each interactive job is equipped with a "priority class" and an actual "priority".

The priorities (and -classes) change according to the behaviour of a job and the main rules are:

1. interaction (= terminal input) implies increasing priority
2. heavy cpu-load (= using entire time slices) implies decreasing priority.

An interactive job starts at the max. priority of the system (= 0). No job will ever exceed the max. priority.

In the following the main algorithms used at job scheduling are listed:

- A. An interactive job is suspended because it asks for terminal input:

```

if priority_class + classgain > 0
  then priority_class: = 0
  else priority_class:=priority_class + classgain;
priority: = priority_class;

```

- B. An interactive job is suspended because the time slice had expired:

```

if priority_class - classloss < minprio
  then abort_job
  else priority_class: = priority_class-classloss;
priority: = priority_class;

```

- C. When the "best" job is to be found in the interactive queue, SOS proceeds like this:

```

job: = queue.first;
while job.priority < max_priority do
begin
  job.priority: = job.priority + priogain;
  put_back_on_queue (job);
  job: = queue.first;
end;

```

thus all interactive jobs stay in the same queue no matter the priority. However, a low priority will imply that the job will be bypassed by other jobs a number of times before activation. During a cpu-bound period an interactive job will thus spend increasing periods of time between its active time slices (in case the whole system has a steady load).

Terminals accessed by SOS jobs may be:

- 1) The primary input/output terminal from which the job was created.
- 2) A number of terminals connected to a terminal group in TEM.
This terminal group must be created by the job itself.

The message flow controlling the access is described in subsections 3.3.1 and 3.3.2. In this section some conventions and limitations concerning terminal access shall be mentioned.

- A. Programs using terminal access under SOS must follow the standard conventions for transfer checking. The most important rule is that input or output messages answered by an empty answer (no data transferred) are repeated.
- B. SOS jobs will never see hard errors on terminals because SOS will automatically disconnect such a terminal. In case the terminal was primary input terminal of a job, the job is aborted. In case the terminal was connected to a multiterminal group, it will become disconnected and the job will receive a special input telling that the terminal is disconnected.
- C. SOS jobs may create at most one terminal group for multiterminal access. This group must have the name of the terminal access module (usually TEM). Jobs may connect a number of terminals to this pool (by creating TEM-links). In case the terminals are connected by the job itself, SOS is not directly involved and there will be no possibility for SOS to check the authorization of the terminal users. This facility may for instance be used for accessing F8000 terminals.
- D. When an SOS job has created a terminal group, SOS may connect terminals to this group. This is done in case a terminal user "logs in" by using the "in" command. If the user is allowed to

be connected to the job (according to the user catalog), SOS will do so and to inform the job of the arrival of a new user, SOS generates a special input line. In a similar way SOS may disconnect terminals using the "out" command. The contents of these special input lines are:

```
login:  <localid.>1>1>32>att<32>32>32><process name><NL>
logout: <localid.>2>2>32>out<32>32>32><NL>
hard er-
ror:    <localid.>2>2>32>hard<32>error<NL>
```

Only TTY-compatible terminals may use the "in" and "out" facilities of SOS.

3.3 SOS - Job Intercommunication

3.3

An SOS job may access terminals, backing storage and eventually a few other kinds of devices. All kinds of access are on the basic level performed by using the "send message-wait answer" functions. The communication with backing storage, for instance, will never be affected by SOS. However, SOS intervenes all communication between an SOS job and its terminal(s), to know precisely the current state of the job.

Thus even though a job "believes" that it communicates with its primary input/output terminal (and eventually TEM) it really communicates with SOS.

Apart from the terminal communication there is a "parent communication" (i.e. jobs send messages to their parent, SOS, when they finish, in case they want documents mounted, if they detect severe internal errors or the like).

3.3.1 Primary Input and Output

3.3.1

The primary input and output communication performed by a job may be visualized like this:

<u>Job</u>	<u>SOS</u>	<u>Terminal (or TEM)</u>
input message ->	stop process	
	input message ->	
	-	
	(other jobs may run)	
	-	
	put job into	<- input answer
	active queue	
	-	
	copy input	
<-	input answer	
	activate job	
	-	
	-	
output message ->	copy output	
	output message ->	
<-	output answer	
	-	
	-	
		<- output answer

Only two message operations are involved:

input:

message: + 0
+ 2
+ 4
+ 6
+ 8
+10
+12
+14

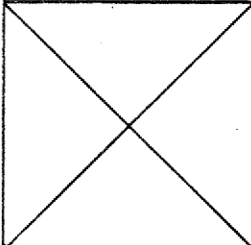
3	mode
first address	
last address	

The input messages may for instance be sent when using the read procedures of ALGOL. The zone used for terminal communication should be opened exactly as to a "real" terminal:

open (z, mode shift 12 + 8, <name of SOS system>, give up)

output:

message: + 0
 + 2
 + 4
 + 6
 + 8
 +10
 +12
 +14

5	mode
first address	
last address	
	

The output messages may for instance be sent, when using the write procedures of ALGOL. The zones used for output should be opened exactly as for input.

3.3.2 Transparent Communication to TEM

3.3.2

SOS jobs may use TEM by sending messages to the TEM process as described in the TEM manual (ref. [4]). All messages to TEM will be sent via SOS and SOS imposes a set of limitations.

1. An SOS job may only create one terminal group.
2. The terminal group created by a job must use the name of the TEM process (usually TEM).
3. The buffer length of SOS will be the max block length of data transfers.
4. No dummy message will be returned to the job after pool creation.

As TEM may spool input and output, the communication job-SOS-TEM looks slightly different from the primary input/output communication:

<u>Job</u>	<u>SOS</u>	<u>TEM</u>
output message ->	output message ->	
	send timer	
	wait first answer	
	(output or timer)	
	-	
	regret timer	<- output answer
<-	output answer	
	-	
input message ->	input message ->	
	send timer	
	wait first answer	
	(input or timer)	
	-	
	if timer answer first	
	then stop job	
	-	
	(other jobs may run)	
	-	
	put job into	<- input answer
	active queue	
	-	
	copy input	
<-	input answer	
	activate job	

The reason for using a timer period (of 50-100 m.sec.) is to avoid swopping a job in case TEM is able to answer within this period (often TEM will need a little time for transferring spooled data).

In the communication between a job and TEM, SOS will offer a special feature. The usual convention is that each block of output sent to TEM must include address information (in case of TTY

compatible terminals, the first 24 bits of a block is interpreted as an address). When communicating via SOS, it is possible to use an implicit addressing mode for output. In this mode no address information is needed in the block. Instead the output block is routed to the terminal from which the job has received its latest input block. (This way of addressing may often be convenient as most terminal communication looks like terminal commands immediately followed by an answer).

To use this facility the outputzone must be opened like this:

```
open (z, 1 shift 18 + mode shift 12 + 8, <name of TEM>, 0);
```

3.3.3 Parent Messages

3.3.3

Jobs may send parent messages to inform the system of job termination, severe errors or to request operator action.

Parent messages all use the following format:

message: + 0

+ 2

+ 4

+ 6

+ 8

+10

+12

+14

function	pattern <5 + wait
integer or text portion	
	-
	-
	-
	-
	-
	-

"function" specifies the operation to be performed. Only even values are allowed.

"pattern" specifies how the parent (SOS) is to display the message (on the system terminal). The "pattern" contains seven bits, one to each of the last seven words of the message (+2,+4.....+14).

When a bit equals one it means that the corresponding word should be displayed as an integer, otherwise the word is displayed as a text portion of 3 characters. Thus bit 1<11 means that the second word of the message is an integer and 1<5 means that the last word of the message is an integer.

"wait" may be zero or one. A zero means that the job wants an answer immediately, one means that the job should not be answered (restarted) until some operation (operator action) is completed.

When receiving a parent message, SOS will perform the following actions depending on the function of the message:

function = 2: finis

The job is aborted. Temporary files are cleared, eventual TEM pools and links are removed. The job process is removed and a finis message is written on the start-up terminal of the job.

function = 4: break

The actions are exactly as for finis. However, the message written on the terminal is different.

function = 14: mount

If the document is already mounted and accessible, the job will become a user of the device and SOS returns the answer immediately. If the document is not mounted, SOS displays the message and stops the job. When the operator has mounted the job he may activate it by using the "start" command.

function = 16: print

Displays the contents of the message according to the general rules. Is the wait bit zero the job will be answered at once, otherwise it is stopped and the operator may reactivate it by using the "start" command.

function = 32: mount special

Treated exactly as "mount" (function = 14)

All other functions are treated exactly like a "print" message
(function = 16).

4. OPERATING GUIDE

4.

The operators tasks in the day to day running of the system comprise the following major points:

- a. start-up
- b. system and job intervention
- c. device support
- d. close down
- e. system failure.

4.1 Start-Up

4.1

During the start-up the system calculates a set of minimum resources on the basis of the trimmed values. This set of resources must be present in order that the run can be succesful, otherwise the run is immediately terminated with a message specifying the minimum set of resources. When started with sufficient resources the resources available for jobs are listed.

@

ATT sos

new sos internal 3 size 50000 buf 30 area 30 perm disc 1000 40
READY

@

ATT s

function 1,2,3,4,5 prog bsos base -8388607 8388605 run
READY

FROM SOS

02.58 SOS: MESSAGE SOS VERSION: 780929 0

02.58 SOS: MESSAGE SOS CHILD RESOURCES

02.58 SOS: MESSAGE SOS INTERNAL 3

02.58 SOS: MESSAGE SOS BUF 16

02.58 SOS: MESSAGE SOS AREA 24

02.58 SOS: MESSAGE SOS SIZE 45568

02.58 SOS: MESSAGE SOS STARTED

Example 16: SOS started as a child of "s".

```

@
ATT s
replace prologue

@
ATT
READY
PROCESS NAME = sos

FROM SOS
PROGRAM NAME = bsos
09.02 SOS: MESSAGE SOS VERSION: 780901 0
09.02 SOS: MESSAGE SOS CHILD RESOURCES
09.02 SOS: MESSAGE SOS INTERNAL 3
09.02 SOS: MESSAGE SOS BUF 147
09.02 SOS: MESSAGE SOS AREA 134
09.02 SOS: MESSAGE SOS SIZE 50688
09.02 SOS: MESSAGE SOS STARTED

```

Example 17: SOS started using "S-replacement".

It should be mentioned here that even though all examples use the system name "SOS", any name of up to 8 characters may be used.

4.2 Operator Intervention

4.2

At system trimming an operator key (an operator password) is defined which may be used at operator intervention. Operator interventions are accepted only when initiated from the system terminal of SOS (the terminal used at SOS start-up).

The operator interventions concern the SOS system (= system intervention) or one or more jobs (= job intervention).

4.2.1 System Intervention

4.2.1

By using the system intervention commands the operator will be able to change the state of the system. The possible commands are:

$$\left\{ \begin{array}{l} \text{lock} \\ \text{open} \end{array} \right\} \langle \text{operator key} \rangle$$

(These commands are only accepted when typed on the system terminal of SOS).

The "lock" command will make SOS refuse all attempts to create jobs or to connect terminals to multiterminal jobs. The command may be used for draining the system. As a consequence SOS will write a message on the system terminal when the last job leaves, telling the operator that the system is empty.

```
@
ATT sos
>lock opr
16.56 SOS: READY
```

Example 18: Operator intervention.

The "open" command is the opposite of the "lock" command. After "open" job creation and terminal connection will be accepted.

4.2.2 Job Intervention

4.2.2

The job intervention commands (described in chapter 2) may in a changed version be used as operator commands.

In this version the syntax is:

$$\left\{ \begin{array}{l} \text{stop} \\ \text{start} \\ \text{break} \\ \text{kill} \end{array} \right\} \langle \text{operator key} \rangle \left\{ \begin{array}{l} \langle \text{jobname} \rangle \\ \text{all} \end{array} \right\}$$

In case the "all" option is used, all jobs enrolled will be subject to the action stated.

The "stop" command will suspend the job(s) in question in such a way that the execution may be resumed.

The "start" command will activate suspended jobs. The suspension may for instance be a "pause" message requesting an operator action.

The "break" and "kill" commands abort the job(s) in question. At "break" the job will be allowed to write an error message before removal.

```
@
ATT sos

>kill opr all
16.59 SOS: READY
```

Example 19: Clear system.

4.2.3 Device Support

4.2.3

Usually SOS jobs will not use equipment that requires operator support. There will be a need, however, for users to make safety copies on flexible disc, magnetic tape or the like. Jobs using that kind of equipment may request documents to get mounted. These requests will be displayed on the system terminal of SOS. When the document is mounted, the operator must name the device (with the name of the document mounted) and then restart the job. The naming is performed by the "call" command:

```
call <device number> <document name>
```

(the "call" command may be immediately followed by the "start" command).

```
16.58 SOS: PAUSE RC MOUNT MT
```

```
@
ATT sos

>call 10 mt start opr rc
16.59 SOS: READY
```

Example 20: Tape mounting.

4.3 Close Down

4.3

The SOS system is closed by using the "halt" command:

```
halt <operator key>
```

The run will terminate immediately without removing active jobs (remember that the system may be drained before closing). In case the system generates testoutput "halt" will close the testoutput file. After closing SOS, the system (the SOS process) may be removed.

```
@
ATT sos

>lock opr
16.54 SOS: MESSAGE SOS SYSTEM EMPTY
16.54 SOS: READY

@
ATT sos
>halt opr
16.55 SOS: PAUSE SOS SYSTEM CLOSED:
```

Example 21: Drain system and close down.

4.4 How to Handle System Failure

4.4

During the run the system may break down in one of the following two ways:

1. An internal program error or a transport error from the program area "bsos" may cause the system to break down and the following error message will be printed on the system terminal of SOS.

```
pause sos ***fault
```

(Please notice that if this terminal is reserved by other processes (e.g. by login to BOSS) it will not be possible for SOS to print these messages).

2. A hard error in the swop area makes continued running impossible and the system stops after having printed the message:

```
message sos status <statusword> swpsos
```

In all error situations one should, if the system has been trimmed with 'testoutput', move this from the test area (e.g. TSTSOS) to a work area, from which the TRACE-program can print it for a further analysis.

The TRACE program is automatically generated at the installation of the system. The program is called as follows:

```
trace <testarea>.<segments>
```

<testarea> is the name of the area, from which the test output is to be printed (the work area the test output has been moved to, or the test area itself).

<segments> are the maximum number of segments to be analyzed. TRACE always finds the latest generated segment, and counts the number of segments backwards from there. <segments> are automatically cut to the size of the area, if something larger has been specified.

```
@
ATT s

proc sos remove prog fp run

READY
TO SOS
o lp

trace tstsos.10000

o c
@
ATT s

proc sos remove
```

Example 22: Printing testoutput.

(Once again, please notice that the names used (SOS, TSTSOS and SWPSOS) are only examples, as other names, consisting of up to 8 characters, as well might be used, allowing identical systems each with its own name to run simultaneously.

5. INSTALLATION NOTES

5.

This chapter concerns the subjects that are relevant for persons who administers the resources of an installation and for persons who are actually going to install an SOS system on a computer of the RC4000 or RC8000 series.

5.1 System Distribution

5.1

The SOS system is usually distributed as a magnetic tape containing the files below:

- 0: label
- 1: soshelp (is used as an auxiliary file when generating an SOS system; contains the files "sostrim", "sossave" and "sosload")
- 2: trcmol (compiler for generating "bsos", "cleartemp" and "prologue")
- 3: tsoscat (a simple user catalog)
- 4: tsos (system program text)
- 5: tcleartemp (program for clearing temporary files when a job terminates)
- 6: tupsoscat (program for generating, updating and listing the user catalog)
- 7: ttrace (program for analyzing test output)
- 8: tsostest (multiterminal program for simple system testing)
- 9: tprologue (program used for starting a system as "S-replacement")
- 10: tdisplay (program displaying all jobs actually running under SOS)
- 11: tnews (test file to be written on terminals running the "hotnews" job defined in "tsoscat")

All files of a system tape may be automatically loaded to backing storage like this:

```
sosdoc = set 1 disc3
soshelp = set mto sostape 0 1
i soshelp
i sosload
```

In case no "sosdoc" is specified the files will be loaded primarily on the system disc.

On the other hand a system tape may be generated from an SOS system on disc by:

```
sosdoc = set mto sostape
i soshelp
i sossave
```

and a standard system may be generated from tape or disc by using this set of commands:

```
soshelp = set mto sostape 0 1      ; only for tape
i soshelp
i sostrim
```

5.2 System Trimming

5.2

The quantities to be defined when the system is trimmed, fall into two groups:

- a. System constants concerning the strategy of execution, time slice, test output, the type of machine and the buffer length for the applications terminal I/O.
- b. The minimum of resources available for the applications and the terminal users. E.g. 1) the minimal core size for applications, 2) the minimal number of simultaneous applications etc.

Regarding job resources, the system has been designed to calculate on the basis of its start up resources, how many jobs it will be able to process simultaneously. If this number is smaller than the minimum specified in the trimming, or if the remaining pool of resources (buffers, areas etc.) is smaller than what is specified in the trimming, the system will stop after the initialization with an error message.

In the following all constants that may be changed in the trimming are mentioned.

"optionid"

At start-up a constant showing the date of the SOS version will be listed together with this constant. At each trimming this constant should be changed to show the date of the trimming (e.g. 780901). The standard value is 0 indicating that "standard trimming" is used.

"rc"

This constant defines the target machine to which the system is trimmed. The only values accepted are 8000 or 4000.

"minusers"

The minimal number of jobs that may be enrolled simultaneously (i.e. the number of internal processes allocated to SOS).

"commdusers"

Even when the maximal number of jobs are running there will be a need for handling terminals, performing operator commands, login commands or the like. "commdusers" defines the minimum of terminals that will be able to communicate with SOS without having created any job.

"minbufs"

Defines the minimal set of message buffers in the pool of resources that may be allocated to jobs.

"minareas"

Defines the minimal set of area processes in the pool of resources that may be allocated to jobs.

"minsize"

The free size of primary store in an SOS system depends on the size of the SOS process as defined at start-up. During initialization SOS will check that the "free size" will be equal to or exceed "minsize". The standard value (12800) is sufficient for running most of the utility programs and compilers.

"buf1"

Defines the size of the I/O buffer used for communicating data between terminals and jobs. (Communication via TEM will pass the same buffer - and use the same buffer limit). The standard value (104) matches the terminal buffer size as defined in the standard I/O system.

"timeslice"

SOS allocates computing time in time slices. When a job is swopped into primary store and activated, it will at the latest be suspended after the expiration of a time slice. Then the state and priority of all jobs enrolled will be evaluated and the "winner" will get the next time slice. A "large" time slice will decrease overhead and increase response time variations. The standard value (3 seconds) will often do.

"cpulimit"

Interactive jobs running under SOS will not be allowed to cycle indefinitely. At most it will be allocated "cpulimit"/"classloss" timeslices before being removed from the system. In case the priority of the job was lower than the maximum priority when the cycle started, then it will be allocated less than "cpulimit"/"classloss" timeslices before removal. (For further investigations see chapter 3).

Please notice: $0 < ("cpulimit" + "classloss") < 2048$.

"classloss", "classgain", "priogain".

These three constants all concern the scheduling strategy of SOS. This strategy is explained in detail in chapter 3. Here some rules of thumb shall be stated:

- a. $1 < \text{<constant>} < 2048$
- b. when "classloss" is large, jobs will rapidly be removed by "time exceeded";
- c. when "classgain" is large, jobs will quickly forget that they have had a cpu-bound period (a period of low priority);
- d. when "priogain" is large, the response time will (statistically) be proportional to the computing time, while a small value of "priogain" will give response times that increase more than proportional to the computing time.

"testsegmnts"

The SOS system may currently generate testoutput for maintenance purposes. The testoutput is cyclically written into a testoutput file. "testsegmnts" defines the size of this area. Obviously the size of the testoutput area is proportional to the period of time that may be "traced". If "testsegmnts" equals "0" no testoutput will be generated. It is recommended that testoutput is generated at least during some period after the first installation of the system. (The testoutput is necessary in case you want the RC maintenance staff to analyze and solve your problems. However, you should not waste your time trying to analyze the testoutput yourself).

"conditions"

This constant is a bitpattern defining the reaction on time exceeded (see "cpulimit") and break. The standard value is 2'000000. The bits used are:

2'000001 = abort job at time exceeded. When this bit is zero, an online job may run for ever.

2'000010 = abort job after break command. When this bit is zero, a user break or operator break will work as "stop load start" when the terminal is running a job without jobfile.

"oprkey"

Defines a text used as operator password. Default is "opr".

This text must be nonempty.

"swopdoc", "testdoc"

In case these two texts are empty, the swop area and the test-output area are usually placed on the system disc (they are actually placed on the first disc on which the SOS process has got backing storage resources). To smooth disc load it may be reasonable to place two areas on some other disc than the system disc.

The system trimming is actually done by means of the file "sostrim" (see appendix F) which contains the standard trimming plus commands for generating the trimmed program version together with some utility and test programs.

The individual trimming consists of changing some values in "sostrim" in case standard values do not apply. This is done by simple editing e.g.:

```
xtrim = edit sofrim
EDIT BEGIN
1./optionid:=/, r/0/780901/,
1./minsize:=/, r/12800/20000/,
1./timeslice:=/,r/3/5/,
1./testsegmnts:=/,r/42/168/,
f
EDIT END.
```

Example 23: Individual trimming.

The commands stated in the file "sostrim" (and here "xtrim") will generate an SOS system like this:

- a. load a compiler for translating SOS system
- b. generate SOS system program: "bsos"
- c. generate catalog cleaning program: "cleartemp"
- d. generate user catalog program: "upsoscat"

- e. generate testoutput analyzing program: "trace"
- f. generate S-replacement loader: "prologue"

In case a file with the name "soscat" already exists, the system generation will stop here. If no "soscat" exists, it is assumed that SOS has not been running on the installation before and therefore a set of testprograms and testfiles is generated to facilitate a system test. The generation proceeds like this:

- g. generate a simple user catalog: "soscat"
- h. generate a multiterminal program: "sostest"
- i. generate a display program: "bdisplay"
- j. generate a news-file: "tnews".

@

ATT s
all sos run
READY

TO SOS
soshelp = set mto sostape 0 1
i soshelp

FROM S
PAUSE SOS MOUNT SOSTAPE
@
ATT s
call 10 sostape start
READY

FROM SOS
xtrim = edit sostrim
EDIT BEGIN.
1./optionid/, r/0/780901/,
1./minsize/, r/12800/20000/,
f
EDIT END.
i xtrim

EDIT BEGIN.
EDIT END.

TRCMOL
1:(RCMOL=SET 1 DISC
1:RCMOL=ALGOL
1:SCOPE USER RCMOL
1:END
1:BEGIN

3068:END

ALGOL END 168

XSOS D.781003.1610

RCMOL/011 D.781003.1616
TRANSLATION TIME = 199.85 SEC
CORE AREA CLAIM = 12246 BYTES
DISC AREA CLAIM = 24 SEGMENTS

TCLEARTEMP D.781003.1615

RCMOL/011 D.781003.1616
TRANSLATION TIME = 8.34 SEC
CORE AREA CLAIM = 504 BYTES
DISC AREA CLAIM = 1 SEGMENTS

END 125

(to be continued)

(continued)

TUPSOSCAT

1:BEGIN

2013:END;

ALGOL END 97

TTRACE

1:;

TRACE

1:;

1:;

1:; PROGRAM FOR ANALYZING TESTOUTPUT

1:

1:

1:BEGIN

174:END

ALGOL END 34

TPROLOGUE

RCMOL/011 D.781003.1617

TRANSLATION TIME = 14.26 SEC

CORE AREA CLAIM = 996 BYTES

DISC AREA CLAIM = 2 SEGMENTS

END 122

END 59

TDISPLAY

1:BEGIN

53:END

ALGOL END 29

SYSTEM GENERATION COMPLETED

Example 24: System installing and trimming.

5.3 User Catalog

5.3

Any job executed under SOS must be described in the user catalog. The user catalog contains information about resource demands, scope (file access), password, start-up commands and in case of a multiterminal job, descriptions of terminal users who are allowed to login to this job.

The user catalog is created and updated by the program "upsoscat". This program may list the actual contents of a user

catalog in such a way that the listing may be used as input for generating a new catalog. As users may change their passwords, it is not convenient to generate a changed catalog from an edited version of the original catalog text. Instead a new catalog may be generated without destroying actual passwords, by using an edited version of an actual catalog listing.

The user catalog consists of a set of job (process) descriptions.

5.3.1 Data per Job (Process)

5.3.1

The user catalog must contain the following information per process:

process name	: max. 8 characters
buffers	} : the process' demands on buffers and areas
areas	
bases	
	: the standard-, user- and max (project) base of the process
password	: max. 11 characters
minsize	: the minimum size acceptable for the process
maxsize	: the max size used for the job (even though SOS may have room for more)
FP-commands	: max. 59 characters are executed when the job is created. Can be used to start-up an application.
bs-claims	: device-name (max. 11 characters) plus entries and segments for key0, key1, key2 and key3. Max. 12 units.
terminals	: external identification (max. 11 characters), local identification (max. 3 characters), password (max. 11 characters),

input buffering (max.no of input buffers
spooled by TEM)
timercount (max.no of timeout periods
expired before TEM returns an answer).

The process name must identify the job unambiguously.

An arbitrary number of terminals can be registered under a process. The external id. and the local id. must be unambiguous for terminals under the same process.

Parts of the descriptions may be omitted. The "upsoscat" program will then generate default values. In the following all default values that are not 0 (or nonempty) are listed.

```

buffers          : 4
areas            : 7
maxsize          : 8388608
bs-claims        : disc key0 6 0 key1 0 0 key2 0 0 key3 0 0
terminal input
buffering        : 1
terminal timer
count            : 40

```

At catalog generation it is checked that process names, buffers and areas are given values different from 0.

5.3.2 Creation and Change of the User Catalog

5.3.2

The program "upsoscat", which is used for generating the user catalog, is called as follows:

$$\left\{ \begin{array}{l} \langle \text{newcat} \rangle \\ \text{no} \end{array} \right\} \begin{array}{l} 1 \\ 0 \end{array} \text{ upsoscat } \left\{ \begin{array}{l} \langle \text{input} \rangle \\ \text{no} \end{array} \right\} \begin{array}{l} 1 \\ 0 \end{array} \left\{ \begin{array}{l} \text{oldcat} \\ \left\{ \begin{array}{l} \langle \text{cat} \rangle \\ \text{no} \end{array} \right\} \end{array} \right\} \begin{array}{l} 1 \\ 0 \end{array}$$

$$\left\{ \begin{array}{l} \text{list.} \\ \text{no} \end{array} \right\} \begin{array}{l} 1 \\ 0 \end{array}$$

<code><newcat></code>	:	the name of the new user catalog
<code><input></code>	:	the name of the file containing the input. If this is omitted, input is taken from the lines following the program call
<code><cat></code>	:	the name of the user catalog to be updated
<code><outfile></code>	:	the name of the file in which the contents of the catalog is to be printed.

If the parameter `oldcat` is omitted or `oldcat.no` is stated, a new user catalog is created in `<newcat>`; otherwise `<newcat>` will contain an updated version of `<cat>`.

If `list` is omitted or `list.no` is stated, the new contents of the catalog are not printed; otherwise the contents will be printed in the `<outfile>` in a manner making it possible to use it as input for `upsoscat`.

`<newcat>`: is reserved by the program throughout the run, whereas `<cat>` is only reserved during the period of copying (this is done at the beginning of the run).

`<newcat>`: is extended by the program if necessary. If this is not possible, the program stops with the message 'lookup `<i>`' or 'ch.entr `<i>`'. In case of an error in the input, the OK-bit is set to 'no'. (The program ends by writing 'errors 0').

5.3.2.1 Creation

5.3.2.1

For each process to be created, the data described in subsection 5.3.1 may be stated.

The syntax for input:

$$\{\text{maxprocess} \quad \langle \text{number} \rangle\}_0^1$$

$$\left\{ \begin{array}{l} \text{process } \langle \text{proc-name} \rangle \\ \{\text{buf } \langle \text{buf} \rangle\}_0^1 \\ \{\text{area } \langle \text{area} \rangle\}_0^1 \\ \text{stdbase } \langle \text{number} \rangle \langle \text{number} \rangle \\ \text{userbase } \langle \text{number} \rangle \langle \text{number} \rangle \\ \text{maxbase } \langle \text{number} \rangle \langle \text{number} \rangle \\ \{\text{password " } \{\langle \text{key} \rangle\}_0^1 \text{ "}\}_0^1 \\ \{\text{minsize } \langle \text{number} \rangle\}_0^1 \\ \{\text{maxsize } \langle \text{number} \rangle\}_0^1 \\ \\ \{\text{fp " } \{\langle \text{text} \rangle\}_0^1 \text{ "}\}_0^1 \\ \\ \left\{ \begin{array}{llll} \text{bs } \langle \text{name} \rangle & \text{key0} & \langle \text{entr} \rangle & \langle \text{segm} \rangle \\ & \text{key1} & \langle \text{entr} \rangle & \langle \text{segm} \rangle \\ & \text{key2} & \langle \text{entr} \rangle & \langle \text{segm} \rangle \\ & \text{key3} & \langle \text{entr} \rangle & \langle \text{segm} \rangle \end{array} \right\}_0^{12} \\ \\ \{\text{term } \langle \text{name} \rangle \text{ " } \{\langle \text{local id.} \rangle\}_0^1 \text{ " " } \{\langle \text{key} \rangle\}_0^1 \text{ " } \\ \quad \{\langle \text{bufs} \rangle \quad \{\langle \text{timeouts} \rangle\}_0^1\}_0^1 \}_0^\infty \\ \\ \{\text{end}\}_0^1 \end{array} \right\} \text{max.}$$

$\langle \text{proc-name} \rangle$: max. 8 characters, letters or digits.
 $\langle \text{name} \rangle$: max. 11 characters, letters or digits.
 $\langle \text{buf} \rangle$, $\langle \text{area} \rangle$, $\langle \text{entr} \rangle$, $\langle \text{segm} \rangle$: non-negative integers.
 $\langle \text{text} \rangle$: max. 59 characters, all characters except " are allowed.

 $\langle \text{local.id} \rangle$: max. 3 characters, all characters except " are allowed.
 $\langle \text{key} \rangle$: max. 11 characters, all characters except " are allowed.

Note! The "term" option must not be followed by any other option within a process description. (When used, the terminal should be the last part of the process description).

On the basis of <number> after maxprocess it is calculated how many processes <max>, there must be room for in the user catalog. Max is the smallest number, which is a multiple of 50 and which is bigger than or equals <number>.

If maxprocess omitted max is set to 50.

The catalog bases are defined by right and left limits for the base intervals.

The parameter 'end' need not be included if input is specified in the program call.

If the demands described in subsection 5.3.1 are not met, the process is not registered. Errors in the terminal parameters will only effect that the terminal is not registered.

The catalog is created directly in <newcat>. After being created, <newcat> will always contain a correct user catalog, but if there have been errors in the input, the catalog will not correspond to what was wanted. By errors it is recommended to make a rerun instead of updating <newcat>.

5.3.2.2 Change

5.3.2.2

There are three types of updating which concern a process: correction (cprocess), creation (iprocess) and deletion (dprocess).

When correcting, the information to be changed plus its new value must be stated. If all the entries and segments of a unit are zero, the unit is deleted. The terminal can be created and deleted (with the parameters term and dterm). The terminal corrections must be the last changes, which are specified for a process.

The syntax for input is:

{	<pre> cprocess <proc-name> {buf <buf>}₀¹ {area <area>}₀¹ {stdbase <number> <number>}₀¹ {userbase <number> <number>}₀¹ {maxbase <number> <number>}₀¹ {password " {<key>}₀¹ " }₀¹ {minsize <number>}₀¹ {maxsize <number>}₀¹ {fp " {<text>}₀¹ " }₀¹ { bs <name> key0 <entr> <segm> key1 <entr> <segm> key2 <entr> <segm> key3 <entr> <segm> } </pre>	} <div style="display: inline-block; vertical-align: middle; margin-left: 10px;"> ¹² 0 </div>
{	<pre> term <name> " {<local id.>}₀¹ " " {<key>}₀¹ " {<number>}₀¹ {<number>}₀¹ }₀[∞] dterm <name> " {<local id.>}₀¹ " }₀[∞] </pre>	}
{	<pre> iprocess <proc-name> buf <buf> . . . (as for catalog creation) . . </pre>	} <div style="display: inline-block; vertical-align: middle; margin-left: 10px;"> max 0 </div>
{	<pre> dprocess <proc-name> </pre>	}
	<div style="display: inline-block; vertical-align: middle;"> [∞] 0 </div>	
{	<pre> end </pre>	}
	<div style="display: inline-block; vertical-align: middle;"> ¹ 0 </div>	

See the note about "term" in subsection 5.3.2.1.

The changes of <cat> are made in a temporary file.

After the updating is finished, this file is copied to <newcat>.

As when created, <newcat> will contain a correct user catalog, but in case of errors it is recommended to make a rerun instead of updating newcat.

5.3.2.3 Listing

5.3.2.3

When the creating/change processes are completed, the contents of <newcat> are listed as described for the input syntax. If this listing is used as input in a catalog creation, the new catalog will become identical with the one listed from <newcat>.

If a listing, without updating, of an already existing catalog is wanted, the program call below is used:

```
upsoscat oldcat. <cat> list.<outfile>
end
```

5.3.3 Resources Needed for Creating a Catalog

5.3.3

After creation the user catalog will contain index segments and process segments. There will be 1 index segment for each 50 processes in the catalog. Each process will occupy one segment containing the process description and up to 5 terminal descriptions. Processes with more than 5 terminal descriptions will occupy one more segment per 19 terminals (exceeding the first 5).

At updating, the size of the user catalog may increase. The catalog can be compressed by making a printout of the catalog and use this in a re-creation process.

```

upsoscat oldcat.soscat list.out
end
END 59
clear project soscat
soscat = set 1
soscat = upsoscat out
END 55
scope project soscat

```

Example 25: Compressing user catalog.

When creating a user catalog by means of listing, at least 5 buffers will have to be used, and by updating at least 6 buffers.

5.3.4 Error Messages

5.3.4

In case of an error, an error message plus the input line with the error is printed.

Below the word "parameter" is used covering the input keywords: end, maxprocess, process, dprocess, cprocess, iprocess, buf, area, stdbase, userbase, maxbase, maxsize, minsize, password, fp, bs, key0, key1, key2, key3, term, dterm. The word transaction is used for the 6 first words mentioned above:

<u>Error text</u>	<u>Explanation</u>
line too long	more than 120 characters in an input line. The line is not processed and is not printed.
illegal char	illegal character in an input line. The rest of the line is processed.
buf	illegal buf-value. The processing is continued with the next parameter.
area	illegal area-value. The processing is continued with the next parameter.

stdbase	}	illegal base-value
userbase		The processing is continued with the next
maxbase		parameter.
password		illegal password. The processing is continued with the next parameter.
minsize	}	illegal size value.
maxsize		The processing is continued with the next parameter.
fp		an error in the text. The processing is continued with the next parameter.
device name		as for "name"
bs		illegal bs value. The processing is continued with the next parameter.
bsfull		more than four bs units. The unit is not registered. The processing is continued with the next parameter.
proc-params		incomplete process description.
missing		The process is not registered/detected.
base error		the values of the bases are inconsistent. The process is not registered/detected.
claim error		the internal relationship of entries and segments from key0 - key3 is not correct. The process is not registered/updated.
abnormal end		input ends where further input was expected. The process, which was last processed, is not registered/updated.
trans		an illegal parameter was read where a transaction was expected. The processing continues with the next parameter.

name	illegal name. The processing is continued with the next transaction or parameter dependent on the situation.
proc in cat	the process already exists in the catalog. The processing is continued with the next transaction.
loc id	illegal local identification. The terminal is not registered. The processing is continued with the next parameter.
term-key	illegal user-key. The terminal is not registered. The processing is continued with the next parameter.
cat full	an insertion of a process exceeding the maximum allowed is attempted. By creation the processing is terminated and by updating the processing is continued with the next parameter.
term in cat	a terminal with the same local id already exists at this process.

Apart from these error messages, the errors below may occur, all causing the termination of the program.

lookup i	an error in one of the data areas. Ought not appear.
ch.entry i	the catalog cannot be extended; too few resources or an error in the catalog. Ought not appear.
call 0	an error in the call of the program.
temp cre	work areas cannot be created (too few resources).
newcat i	an error in the <cat> specified in the program call.
oldcat i	an error in the <outfile> specified in the program call.

remove i	an error in connection with the removal of a work area. Ought not appear.
errors 0	is printed after a completed creation, updating, if there has been any errors in the input.

5.4 Test Facilities

5.4

After the installation of an SOS system (at least including SOS and TEM) it may be checked that the system installed really works. As explained in section 5.2 a set of test programs and test files will be loaded in case no "soscat" existed before the generation. A simple system test using a variety of the facilities offered by SOS and TEM is shown below.

This example consists of three terminal logs as it includes a multiterminal test (using the tesprogram "tsostest").

@

ATT sos

>go team pass hobo

16.53 SOS: TEAM ENROLLED

o pp

b=algol tsostest

o c

b

16 53 43: LOGIN: 2, ATT SUB010

16 53 59: LOGIN: 9, ATT SUB011

16 54 09: LOGOUT: 9,OUT

16 54 12: LOGOUT: 2,OUT

END 23

The log of two terminals serviced by the multiterminal job
"team" may look like:

@

ATT sos

>in team nn

16.54 SOS: TERMINAL CONNECTED

FROM TEM

SOS TESTPROGRAM READY

qwaqwaqwaqwaqwaqwaqwaqwaq

TERM = 9 LINE = 1: QWAQWAQWAQWAQWAQWAQWAQWAQ

1234567890

TERM = 9 LINE = 2: 1234567890

0987654321

TERM = 9 LINE = 3: 0987654321

@

ATT sos

>out

TO TEM

FROM SOS

16.54 SOS: TERMINAL DISCONNECTED

@

ATT sos

>in term userb pass b2

16.54 SOS: TERMINAL CONNECTED

FROM TEM

SOS TESTPROGRAM READY

abnsjdgehbsgfa

TERM = 2 LINE = 1: ABNSJDGEHBSGFA

qverthj

TERM = 2 LINE = 2: QWERTHJ

dftyuiop

TERM = 2 LINE = 3: DFTYUIOP

(to be continued)

(continued)

ATT sos

>out
TO TEM

FROM SOS
16.54 SOS: TERMINAL DISCONNECTED

Example 26: System test after installation.

It is obvious that an operating system is not really tested by running some test programs. The only way to make a realistic test is to use the system for normal routine duties. As a consequence, faults may appear from time to time, especially during the first period after installing the system. To be able to remove the errors causing system failure, SOS may produce testoutput (as explained in section 5.2). The RC maintenance staff will have almost no chances of detecting the errors unless the system failure is documented by means of testoutput, so during the first period after the installation the testoutput facility should be switched on.

5.5 Resource Demands

5.5

When installing the SOS system (or generating a new version) a process with the following set of resources will do:

area 6
buf 6
size 60000
work 20 20 300 disc

When running, the SOS system will need a set of resources for its private use (apart from the resources set aside for SOS jobs).

The resource demands may be computed like this:

Primary store: (halfwords)

resident code app	= 4000
test buffer (optional)	= 512
descriptions and I/O buffers	
	= (users + command users) * (bufsize + 36)

Message buffers:

constant consumption:	= 4
varying consumption	
	= (users + command users) * 2

Area processes:

constant consumption	6
----------------------	---

Backing storage segments:

testoutput = (as defined in trimming)	
swop area =	
	(size of a user process)/512 * (users + command users)

The standard trimming of SOS (as defined in the file "sostrim") will have the following demands:

Primary store:

Code:	4000
testbuffer:	512
descriptions and buffers:(3+2)*	
	(104+36)700
	<u>4302</u>
message buffers: 4 + (3+2)*2	<u>14</u>
area processes:	<u>6</u>

It should also be mentioned here that each job created by the "run" command uses one pool and one link in TEM. Apart from this, jobs using multiterminal access via TEM, will need one more pool plus a number of terminal links depending on the max. number of terminals simultaneously "logged in" to the job.

A. REFERENCES

A.

- [1] RCSL No 31-D476:
RC8000 Monitor, Part 1
- [2] RCSL No 31-D477:
RC8000 Monitor, Part 2
- [3] RCSL No 31-D364, 31-D607 and 31D379:
System 3, Utility Programs, Part 1, 2 and 3
- [4] RCSL No 31-D513:
Terminal Access Module (TEM)
- [5] RCSL No 31-D571:
PRIMO (2. edition)

B. SOS COMMANDS

B.

Command	Parameters	Ref.
batch	$\langle \text{jobname} \rangle \left\{ \text{jobfile } \langle \text{filename} \rangle \right\}_0^1 \left\{ \text{pass } \langle \text{password} \rangle \left\{ \text{newpass } \langle \text{password} \rangle \right\}_0^1 \right\}_0^1$	2.2.1
break	$\left\{ \begin{array}{l} \langle \text{operator key} \rangle \left\{ \begin{array}{l} \langle \text{jobname} \rangle \\ \text{all} \end{array} \right\} \\ \langle \text{jobname} \rangle \end{array} \right\}_0^1$	4.2.2 2.2.3
call	$\langle \text{device no} \rangle \langle \text{document name} \rangle$	2.2.4 4.2.3
go	$\langle \text{jobname} \rangle \left\{ \text{jobfile } \langle \text{file name} \rangle \right\}_0^1 \left\{ \text{pass } \langle \text{password} \rangle \left\{ \text{newpass } \langle \text{password} \rangle \right\}_0^1 \right\}_0^1$	2.2.1
halt	$\langle \text{operator key} \rangle$	4.3
in	$\langle \text{jobname} \rangle \langle \text{username} \rangle \left\{ \text{pass } \langle \text{password} \rangle \left\{ \text{newpass } \langle \text{password} \rangle \right\}_0^1 \right\}_0^1$	2.2.2
include	$\langle \text{device no} \rangle$	2.2.4 4.2.3
kill	$\left\{ \begin{array}{l} \langle \text{operator key} \rangle \left\{ \begin{array}{l} \langle \text{jobname} \rangle \\ \text{all} \end{array} \right\} \\ \langle \text{jobname} \rangle \end{array} \right\}_0^1$	4.2.2 2.2.3
lock	$\langle \text{operator key} \rangle$	4.2.1
open	$\langle \text{operator key} \rangle$	4.2.1
out		2.2.2
run	$\langle \text{jobname} \rangle \left\{ \text{jobfile } \langle \text{file name} \rangle \right\}_0^1 \left\{ \text{pass } \langle \text{password} \rangle \left\{ \text{newpass } \langle \text{password} \rangle \right\}_0^1 \right\}_0^1$	2.2.1
start	$\left\{ \begin{array}{l} \text{operator key } \left\{ \begin{array}{l} \langle \text{jobname} \rangle \\ \text{all} \end{array} \right\} \\ \langle \text{jobname} \rangle \end{array} \right\}_0^1$	4.2.2 2.2.3
stop	$\left\{ \begin{array}{l} \langle \text{operator key} \rangle \left\{ \begin{array}{l} \langle \text{jobname} \rangle \\ \text{all} \end{array} \right\} \\ \langle \text{jobname} \rangle \end{array} \right\}_0^1$	4.2.2 2.2.3

C. MESSAGES FROM SOS

C.

SOS may write messages of the following kinds:

- 1) System messages
- 2) User messages
- 3) Parent messages from jobs

In class 1 and 2 there will be normal messages and error messages.

The layout of SOS messages is:

```
<hour> . <minute> SOS: <message>          (normal)
<hour> . <minute> *** SOS: <message>        (error)
```

The system messages may concern start-up or system failure:

System messages at start-up:

area <number>

```
normal: number of areas in job resource pool
error : min. number of areas requested
```

buf <number>

```
normal: number of buffers in job resource pool
error : min. number of buffers requested
```

buflength <size>

```
error: the buffer length must at least be <size>
```

function 1,2,3,4,5

```
error: min. function requested
```

init troubles

```
error: system initialization not ok, run terminated
```

internal <number>

```
normal: max. number of jobs enrolled
```

key <number>

error: at least 1 free protection key must be available
(appears only at RC4000)

size <number>

normal: max. size available for jobs
error : min. size requested

started

normal: telling that the initialization was successful

version: <date of system> <date of options>

normal: indicating the version of the SOS system.

<area name> <integer>

error: written because SOS is not able to read one of its
areas; the following messages may appear:

swpsos <size>

the swoparea could not be created with the size
specified (too few resources on "swopdoc" - see
section 5.2),

tstsos <size>

the testoutput area could not be created with the
size specified (too few resources on "testdoc" -
see section 5.2),

fp <result>

SOS could not read the file processor
(should never occur),

cleartemp <result>

SOS could not read the catalog cleaning program
"cleartemp"; maybe because it does not exist
(result = 3),

soscat <result>

SOS could not read the SOS catalog "soscat";
maybe because it does not exist (result = 3).

System messages at runtime:

fault

error: caused by an internal error in SOS, monitor or hardware (most likely: SOS)

fault 8' <octal status> <program area>

error: transport error concerning program area (disc failure)

status <decimal status> <area name>

error: transport error concerning swop area or testoutput area (disc failure)

User messages:

bad password

error: password not correct (or missing).

bs claims exceeded

error: SOS has not sufficient backing storage resources for creating the job.

call not ok

error: the "calling" of a device has been rejected (cause: the device is reserved, the device does not exist or the like).

command unknown

error: the command typed is not an SOS command.

disconnection not ok

error: the disconnection of a terminal (at "out") has been refused by TEM (should never occur).

forbidden

error: the user is not authorized for using the command in question.

forbidden - system locked

error: a job creation command or an "in" command has been refused because the system is locked (by the operator).

identification illegal

error: the jobname of a job creation command is not included in the user catalog, or the username of an "in" command does not belong to the jobname specified.

include not ok

error: the device inclusion has been rejected (cause: the device does not exist).

jobfile does not exist

error: the jobfile started in a job creation command is not visible from the catalog bases of the job.

no room in primary store

error: the min. size of the job exceeded the max. size available for SOS jobs. (The min. size may be stated in the user catalog).

process creation not ok

error: a job creation has been refused due to resource limitation in SOS.

process unknown

error: a job intervention command has been rejected because the job stated was not found.

ready

normal: indicates that a command has been successfully interpreted.

syntax

error: the command was rejected because of a syntax error (illegal character, missing parameter or the like).

terminal busy

error: a job creation connecting the terminal via TEM was refused, because the terminal was already connected to some pool in TEM.

terminal connected

normal: a message telling that an "in" command has been successfully interpreted.

terminal connection not ok

error: an "in" command has been rejected because TEM refused connecting the terminal (maybe because the actual job has not created a terminal pool).

terminal not connected

error: an "out" command or a job intervention command has been rejected because the terminal was not connected to the actual job.

user conflict

error: a job creation command has been rejected because the job existed in advance or because an existing job was using the same standard base as the new one.

<jobname> enrolled

normal: a message telling that a job creation has succeeded.

<jobname> removed after <cause>

normal: a message telling that the job has been removed from the system. The cause may be:

finis:	normal finis message from job
break:	abnormal termination caused by an internal job error (a "break")
terminal failure:	a hard error has occurred on the primary input terminal
operator break:	the operator has provoked a "parent break" (break 8)

user break:	the user has provoked a "parent break" (break 8)
operator kill:	the operator has killed the job
userkill:	the user has killed the job
time exceeded:	an interactive job has been too cpu-bound (i.e. the priority has decreased and reached the min. priority accepted by the system).

Parent messages from jobs

Jobs using devices like magnetic tape stations or jobs running into severe errors may send "parent messages" to SOS. SOS handles parent messages as described in subsection 3.3.3. Some parent messages are printed on the system terminal. This is done with the following format:

$$\langle \text{hour} \rangle . \langle \text{minute} \rangle \text{ SOS: } \left\{ \begin{array}{l} \text{message} \\ \text{pause} \end{array} \right\} \langle \text{jobname} \rangle \langle \text{contents} \rangle$$

The $\langle \text{contents} \rangle$ is totally specified by the job as explained in subsection 3.3.3.

D. AN EXAMPLE OF A MULTITERMINAL PROGRAM

D.

```

;      *** TSOSTEST ***
;
;
; A SIMPLE TESTPROGRAM FOR TESTING THE SOS SYSTEM
; THE PROGRAM ACTS LIKE THIS
;
;   CREATE TERMINAL POOL
; LOOP:
;   READ AN INPUT LINE FROM A CONNECTED TERMINAL
;   (THIS INPUT LINE STARTS WITH A TERMINAL NUMBER)
;   INCREASE LINECOUNT(TERMINAL NUMBER)
;   WRITE TERMINAL NUMBER
;   WRITE LINECOUNT
;   WRITE CONTENT OF INPUT LINE
;   GOTO LOOP

BEGIN
  ZONE POOLIN,POOLOUT(26,1,STDERROR);
  REAL TIME,R;
  INTEGER I,ACTIVETERMINALS,MAXTERMINALS,CURRTERMINAL;

  INTEGER PROCEDURE CREATEPOOL(Z);
  ZONE Z;
  BEGIN
    INTEGER I;
    INTEGER ARRAY ZIA(1:20),SIA(1:12);
    ZONE ZTEM(1,1,STDERROR);
    OPEN(ZTEM,0,(:TEM:),0);
    GETZONE6(Z,ZIA);
    GETSHARE6(ZTEM,SIA,1);
    SIA(4):=90 SHIFT 12;
    FOR I:=0 STEP 1 UNTIL 3 DO SIA(8+I):=ZIA(2+I);
    SETSHARE6(ZTEM,SIA,1);
    MONITOR(16,ZTEM,1,SIA);
    CREATEPOOL:=IF MONITOR(18,ZTEM,1,SIA) (> 1 THEN -1 ELSE SIA(1);
    CLOSE(ZTEM,TRUE);
  END CREATEPOOL;

  MAXTERMINALS:=10;
  ACTIVETERMINALS:=0;

  BEGIN
    INTEGER I,J;
    INTEGER ARRAY LINEBUF(1:100),LINECOUNT(1:MAXTERMINALS);
    FOR I:=1 STEP 1 UNTIL MAXTERMINALS DO LINECOUNT(I):=0;
    OPEN(POOLIN,8,(:TEM:),0);
    OPEN(POOLOUT,1 SHIFT 18 + 8,(:TEM:),0);
    CREATEPOOL(POOLIN);

    (* READ A LINE AND DISPLAY IT ON CORRESPONDING TERMINAL *)

  LOOP:
    READ(POOLIN,CURRTERMINAL);
    I:=1;
    FOR I:=1 WHILE READCHAR(POOLIN,LINEBUF(I)) (> 8 DO I:=I+1;
    SETPOSITION(POOLIN,0,0);
    LINECOUNT(CURRTERMINAL):=LINECOUNT(CURRTERMINAL)+1;
    IF LINEBUF(1) = 1 THEN
      BEGIN COMMENT LOGIN;
        LINECOUNT(CURRTERMINAL):=0;
        WRITE(POOLOUT,(:SOS TESTPROGRAM READY(10):));
        ACTIVETERMINALS:=ACTIVETERMINALS+1;
        SYSTIME(1,0,TIME);
        SYSTIME(4,TIME,R);
        WRITE(OUT,(:DD DD DD),R);
        WRITE(OUT,(: LOGIN: :),(:DD),CURRTERMINAL,(:,:));
        FOR J:=1 STEP 1 UNTIL I DO OUTCHAR(OUT,LINEBUF(J));
        SETPOSITION(OUT,0,0);
      END ELSE
      IF LINEBUF(1) = 2 THEN
        BEGIN COMMENT LOGOUT;
          ACTIVETERMINALS:=ACTIVETERMINALS-1;
          SYSTIME(1,0,TIME);
          SYSTIME(4,TIME,R);
          WRITE(OUT,(:DD DD DD),R);
          WRITE(OUT,(: LOGOUT: :),
            (:DD),CURRTERMINAL,(:,:));
          FOR J:=1 STEP 1 UNTIL I DO OUTCHAR(OUT,LINEBUF(J));
          SETPOSITION(OUT,0,0);
        END ELSE
        BEGIN
          WRITE(POOLOUT,(: TERM = :),(:DD),CURRTERMINAL,
            (: LINE = :),(:DDD),LINECOUNT(CURRTERMINAL),(:: :));
          FOR J:=1 STEP 1 UNTIL I DO OUTCHAR(POOLOUT,LINEBUF(J));
        END;
        SETPOSITION(POOLOUT,0,0);
        IF ACTIVETERMINALS > 0 THEN GOTO LOOP;
      END;
    END
  END
END

```

E. AN EXAMPLE OF A USER CATALOG

E.

PROCESS RC	BUF 25	AREA 25
STDBASE	810	810
USERBASE	810	819
MAXBASE	800	899
MAXSIZE	60000	
PASSWORD	"NN"	
BS DISC	KEY0 20 500 KEY1 20 500 KEY2 5 300 KEY3 5 300	
PROCESS HOTNEWS	BUF 4	AREA 4
STDBASE	899	899
USERBASE	899	899
MAXBASE	899	899
MAXSIZE	12800	
FP "(O PP		
C=COPY TNEWS		
FINIS)		
"		
BS DISC	KEY0 4 50 KEY1 0 0 KEY2 0 0 KEY3 0 0	
PROCESS DISPLAY		
STDBASE	898	898
USERBASE	898	898
MAXBASE	898	898
MAXSIZE	12800	
FP "BDISPLAY		
FINIS		
"		
BS DISC	KEY0 4 50 KEY1 0 0 KEY2 0 0 KEY3 0 0	
PROCESS RCSAVE	BUF 4	AREA 7
STDBASE	897	897
USERBASE	890	899
MAXBASE	800	899
BS DISC	KEY0 10 150 KEY1 0 0 KEY2 0 0 KEY3 0 0	
PROCESS TEAM	BUF 4	AREA 7
STDBASE	820	820
USERBASE	820	829
MAXBASE	800	899
MINSIZE	30000	
PASSWORD	"H0B0"	
BS DISC	KEY0 6 300 KEY1 1 20 KEY2 1 20 KEY3 1 20	
TERM USERA	" 1,"	"A1"
TERM USERB	" 2,"	"B2"
TERM USERC	" 3,"	"C3"
TERM NN	" 9,"	" "

END

F. THE AUXILIARY FILE "sostrim"

F.

```

;          *** SOSTRIM ***
;
; CONTAINS OPTIONS FOR TRIMMING SOS SYSTEM
; AND COMMANDS FOR AUTOMATIC SYSTEM GENERATION FROM THE SOS SYSTEM TAPE

SOSDUMMYOUT=SET 1

XSOS = EDIT TSOS          ; EDIT OPTIONS INTO PROGRAM TEXT
L./BODY OF INIT/,
L./===TRIMSTART/,
D./===TRIMFINIS/,
I/
! DATE OF OPTIONS          ! OPTIONID  :=  0,

! TARGET MACHINE (RC4000=4000,RC8000=8000)      ! RC          :=  8000,
! MIN. NO OF USER PROCESSES ACTIVE AT THE SAME TIME ! MINUSERS   :=  1,
! MIN. NO OF ENTRIES FOR TERMINALS PERFORMING OS COMMANDS ! COMNDUSERS :=  2,
! MIN. NO OF BUFFERS RESERVED FOR USER PROCESSES      ! MINBUFS    :=  4,
! MIN. NO OF AREAS RESERVED FOR USER PROCESSES        ! MINAREAS   :=  7,
! MIN. CORE SIZE FOR USER PROCESSES (HALFWORDS)       ! MINSIZE    := 12800,
! SIZE OF I-O BUFFER FOR EACH USER PROCESS (HALFWORDS) ! BUFL       := 104,
! LENGTH OF A TIME SLICE (SECONDS)                   ! TIMESLICE  :=  3,
! MAX NO OF TIME SLICES USED IN CPU (NO INPUT)        ! CPULIMIT   := 25,
! LOSS OF PRIORITY CLASS WHEN TIMED OUT               ! CLASSLOSS  :=  1,
! PRIORITY CLASS GAIN AT INPUT (IF CLASS < 0)         ! CLASSGAIN  :=  1,
! PRIORITY GAIN WHEN FIRST IN ACTIVEQUEUE             ! PRIOGAIN   :=  1,
! SIZE OF TESTOUTPUT AREA (SEGMENTS)                  ! TESTSEGMENTS:= 42,

PRINTTEXTS: TEXT(11)

! OPERATOR KEY          ! OPRKEY     := "OPR",
! DOCUMENT FOR SWOPAREA ! SWOPDOC    := "",
! DOCUMENT FOR TEST AREA ! TESTDOC    := "",

./F

```

```

O SOSDUMMYOUT
MODE 1.NO
LOOKUP SOSDOC ; IF (SOSDOC) IS NOT PRESENT
IF OK.NO
MODE 1.YES
O C
IF 1.YES
SOSDOC = SET 1 ; THEN CREATE IT PREFERABLY ON DISC
RCMOL = ALGOL TRCMOL ; THEN GENERATE A TEMPORARY ONE

BSOS = ENTRY 20 SOSDOC
BSOS = RCMOL XSOS ; TRANSLATE TRIMMED PROGRAM TEXT

CLEARTEMP = ENTRY 10 SOSDOC
CLEARTEMP = RCMOL TCLEARTEMP ; TRANSLATE CATALOG CLEANING PROGRAM

UPSOSCAT = ENTRY 100 SOSDOC
UPSOSCAT = ALGOL TUPSOSCAT ; TRANSLATE PROGRAM FOR GENERATING SOS USER CATALOG

TRACE = ENTRY 50 SOSDOC
TRACE = ALGOL TTRACE ; TRANSLATE PROGRAM FOR ANALYSING TESTOUTPUT

PROLOGUE = ENTRY 4 SOSDOC
PROLOGUE = RCMOL TPROLOGUE ; GENERATE LOADER FOR S-REPLACEMENT
PROLOGUE = CHANGEENTRY PROLOGUE PROLOGUE PROLOGUE PROLOGUE PROLOGUE 8.PROLOGUE PROLOGUE

O SOSDUMMYOUT
MODE 1.NO
LOOKUP SOSCAT ; IF SOSCAT IS NOT PRESENT
IF OK.NO
MODE 1.YES
O C
IF 1.YES ; THEN
( SOSCAT = ENTRY 1 SOSDOC ; BEGIN
  SOSCAT = UPSOSCAT TSOSCAT ; GENERATE AN EXPERIMENTAL USER CATALOG AND:
  XSOSTEST = ENTRY 50 SOSDOC ;
  XSOSTEST = MOVE TSOSTEST ; A SIMPLE SYSTEM TEST PROGRAM
  BDISPLAY = ENTRY 20 SOSDOC ;
  BDISPLAY = ALGOL TDISPLAY ; A PROGRAM DISPLAYING RUNNING SOS-JOBS
  XNEWS = ENTRY 1 SOSDOC ;
  XNEWS = MOVE TNEWS ; AN EXAMPLE OF A "NEWS-FILE"
  CLEAR TEMP TSOSTEST TNEWS ;
  RENAME XSOSTEST.TSOSTEST ;
  RENAME XNEWS.TNEWS ;
  SCOPE USER BDISPLAY TNEWS ;
  SCOPE USER SOSCAT TSOSTEST ; END
)

SCOPE USER BSOS UPSOSCAT TRACE CLEARTEMP

O SOSDUMMYOUT
CLEAR TEMP XSOS TRCMOL RCMOL SOSTRIM TSOSCAT TSOS TUPSOSCAT TTRACE,
TSOSTEST SOSLOAD SOSSAVE SOSLIST TCLEARTEMP TPROLOGUE TDISPLAY TNEWS

O C
CLEAR TEMP SOSDUMMYOUT

MESSAGE SYSTEM GENERATION COMPLETED

```


G. THREE VERSIONS OF A MASTER MIND PROGRAM

G.

G.1 Simple Single Terminal Version

G.1

```

BEGIN
  INTEGER ARRAY SOLUTION,GUESS(1:4);
  INTEGER I,J,X,DIGITOK,DIGITINCLUDED;

  SETCOMBINATION:
  FOR I:=1 STEP 1 UNTIL 4 DO
  BEGIN
    RANDOM(X);
    SOLUTION(I):=X MOD 10;
    FOR J:=1 STEP 1 UNTIL I DO
      IF SOLUTION(I) = SOLUTION(J) AND I (>) J THEN I:=I-1;
    END;
    WRITE(OUT,(:MASTER MIND PROGRAM READY(10):));
  END;

  NEXT:
  SETPOSITION(OUT,0,0);
  WRITE(OUT,(: :));
  FOR I:=1 STEP 1 UNTIL 4 DO READ(IN,GUESS(I));
  DIGITOK:=DIGITINCLUDED:=0;
  FOR I:=1 STEP 1 UNTIL 4 DO
  BEGIN
    FOR J:=1 STEP 1 UNTIL 4 DO
      IF GUESS(I) = SOLUTION(J) THEN
      BEGIN
        IF I = J THEN DIGITOK:=DIGITOK+1
        ELSE DIGITINCLUDED:=DIGITINCLUDED+1;
      END;
    END;
    WRITE(OUT,(<<DD>>,GUESS(I)));
  END;
  WRITE(OUT,(: =) :),
    FALSE ADD 43,DIGITOK,
    FALSE ADD 32,4-DIGITOK,
    FALSE ADD 45,DIGITINCLUDED,
    FALSE ADD 10,1);

  IF DIGITOK < 4 THEN GOTO NEXT ELSE
  BEGIN
    WRITE(OUT,(:YOU GOT IT !(10):));
    GOTO SETCOMBINATION;
  END;
END;

```

G.2 Single Terminal Version with Input Checking

G.2

```

BEGIN
  INTEGER ARRAY SOLUTION,GUESS(1:4);
  INTEGER I,J,X,N,M,CLASS,CHARACTER,DIGITOK,DIGITINCLUDED;

SETCOMBINATION:
  FOR I:=1 STEP 1 UNTIL 4 DO
  BEGIN
    RANDOM(X);
    SOLUTION(I):=X MOD 10;
    FOR J:=1 STEP 1 UNTIL I DO
      IF SOLUTION(I) = SOLUTION(J) AND I <> J THEN I:=I-1;
    END;
    WRITE(OUT,(:MASTER MIND PROGRAM READY(10):));
  END;

NEXT:
  SETPOSITION(OUT,0,0);
  I:=0;
  FOR CLASS:=READCHAR(IN,CHARACTER) WHILE CLASS <> 8 AND I < 4 DO
  BEGIN (* ANALYZE ALL CHARACTERS UP TO (NL) *)
    IF CLASS = 2 (* DIGIT *) THEN
      BEGIN
        I:=I+1;
        GUESS(I):=CHARACTER-48;
      END ELSE
        IF CHARACTER <> 32 (* SPACE *) THEN
          BEGIN
            WRITE(OUT,(:***SYNTAX ERROR(10)TRY AGAIN(10):));
            GOTO NEXT;
          END;
        END;
      IF I = 0 THEN GOTO NEXT;
      IF I < 4 THEN
        BEGIN
          WRITE(OUT,(:***GUESS NOT COMPLETE(10)TRY AGAIN(10):));
          GOTO NEXT;
        END;
        FOR N:=2 STEP 1 UNTIL 4 DO
          FOR M:=1 STEP 1 UNTIL N-1 DO
            IF GUESS(N) = GUESS(M) THEN
              BEGIN
                WRITE(OUT,(:***DIGIT DUPLICATED(10)TRY AGAIN(10):));
                GOTO NEXT;
              END;
            END;
          WRITE(OUT,(:
                                :));
          DIGITOK:=DIGITINCLUDED:=0;
          FOR I:=1 STEP 1 UNTIL 4 DO
            BEGIN
              FOR J:=1 STEP 1 UNTIL 4 DO
                IF GUESS(I) = SOLUTION(J) THEN
                  BEGIN
                    IF I = J THEN DIGITOK:=DIGITOK+1
                      ELSE DIGITINCLUDED:=DIGITINCLUDED+1;
                  END;
                END;
              WRITE(OUT,(:DD),GUESS(I));
            END;
          END;
          WRITE(OUT,(:
                    =)
                    :),
                FALSE ADD 43,DIGITOK,
                FALSE ADD 32,4-DIGITOK,
                FALSE ADD 45,DIGITINCLUDED,
                FALSE ADD 10,1);

          IF DIGITOK < 4 THEN GOTO NEXT ELSE
            BEGIN
              WRITE(OUT,(:YOU GOT IT !(10):));
              GOTO SETCOMBINATION;
            END;
          END;
        END;

```

G.3 Multiterminal Version with Input Checking

G.3

```

BEGIN
  INTEGER TERMNO;
  ZONE ZIN,ZOUT(26,1,STDERROR);
  INTEGER ACTIVETERMINALS,FIRSTCHAR;
  ACTIVETERMINALS:=0;
  OPEN(ZIN,8,(:ITEM:),0);
  OPEN(ZOUT,8,(:TEM:),0);

  CENTRALWAIT:
  SETPOSITION(ZIN,0,0);
  READ(ZIN,TERMNO);
  READCHAR(ZIN,FIRSTCHAR);
  IF FIRSTCHAR = 2 THEN
    BEGIN (* LOGOUT *)
      ACTIVETERMINALS:=ACTIVETERMINALS-1;
      IF ACTIVETERMINALS > 0 THEN GOTO CENTRALWAIT ELSE GOTO STOPPROGRAM;
    END ELSE
    IF FIRSTCHAR = 1 THEN
      BEGIN (* LOGIN *)
        ACTIVETERMINALS:=ACTIVETERMINALS+1;
      END ELSE REPEATCHAR(ZIN);
    WRITE(ZOUT,(:DD:),TERMNO,(:,:));

  BEGIN CONTEXT(TERMNO,10,3);
    INTEGER ARRAY SOLUTION,GUESS(1:4);
    INTEGER I,J,X,N,M,CLASS,CHARACTER,DIGITOK,DIGITINCLUDED;
    CONTINUE;

  SETCOMBINATION:
  FOR I:=1 STEP 1 UNTIL 4 DO
    BEGIN
      RANDOM(X);
      SOLUTION(I):=X MOD 10;
      FOR J:=1 STEP 1 UNTIL I DO
        IF SOLUTION(I) = SOLUTION(J) AND I <> J THEN I:=I-1;
      END;
      WRITE(ZOUT,(:MASTER MIND PROGRAM READY(10):));
    END;

  NEXT:
  SETPOSITION(ZOUT,0,0);
  EXIT(CENTRALWAIT);
  I:=0;
  FOR CLASS:=READCHAR(ZIN,CHARACTER) WHILE CLASS <> 8 AND I < 4 DO
    BEGIN (* ANALYZE ALL CHARACTERS UP TO (NL) *)
      IF CLASS = 2 (* DIGIT *) THEN
        BEGIN
          I:=I+1;
          GUESS(I):=CHARACTER-48;
        END ELSE
          IF CHARACTER <> 32 (* SPACE *) THEN
            BEGIN
              WRITE(ZOUT,(:***SYNTAX ERROR(10)TRY AGAIN(10):));
              GOTO NEXT;
            END;
          END;
        IF I = 0 THEN GOTO NEXT;
        IF I < 4 THEN
          BEGIN
            WRITE(ZOUT,(:***GUESS NOT COMPLETE(10)TRY AGAIN(10):));
            GOTO NEXT;
          END;
        FOR N:=2 STEP 1 UNTIL 4 DO
          FOR M:=1 STEP 1 UNTIL N-1 DO
            IF GUESS(N) = GUESS(M) THEN
              BEGIN
                WRITE(ZOUT,(:***DIGIT DUPLICATED(10)TRY AGAIN(10):));
                GOTO NEXT;
              END;
            WRITE(ZOUT,(:
                                :));
            DIGITOK:=DIGITINCLUDED:=0;
            FOR I:=1 STEP 1 UNTIL 4 DO
              BEGIN
                FOR J:=1 STEP 1 UNTIL 4 DO
                  IF GUESS(I) = SOLUTION(J) THEN
                    BEGIN
                      IF I = J THEN DIGITOK:=DIGITOK+1
                        ELSE DIGITINCLUDED:=DIGITINCLUDED+1;
                    END;
                  WRITE(ZOUT,(:DD:),GUESS(I));
                END;
                WRITE(ZOUT,(: =) :),
                  FALSE ADD 43,DIGITOK,
                  FALSE ADD 32,4-DIGITOK,
                  FALSE ADD 45,DIGITINCLUDED,
                  FALSE ADD 10,1);
                IF DIGITOK < 4 THEN GOTO NEXT ELSE
                  BEGIN
                    WRITE(ZOUT,(:YOU GOT IT !(10):));
                    GOTO SETCOMBINATION;
                  END;
            END;
          STOPPROGRAM;
        END

```

H. INDEX

H.

H.1 Survey of Examples

H.1

Example 1: Job creation	4
Example 2: Executing utility program	5
Example 3: Creating a text file	6
Example 4: Executing utility programs	7
Example 5: Program translation	7
Example 6: Text editing	7
Example 7: Execution of interactive program	8
Example 8: Job intervention	9
Example 9: Job termination	9
Example 10: Running a multiterminal program	10
Example 11: Automatic program activation	11
Example 12: Running a batch job from a terminal	12
Example 13: Creating a job file	12
Example 14: Running a job using a job file	13
Example 15: Job file routing job output to printer	13
Example 16: SOS started as a child of "s"	33
Example 17: SOS started using "S-replacement"	34
Example 18: Operator intervention	35
Example 19: Clear system	36
Example 20: Tape mounting	36
Example 21: Drain system and close down	37
Example 22: Printing testoutput	38
Example 23: Individual trimming	45
Example 24: System installing and trimming	47
Example 25: Compressing user catalog	56
Example 26: System test after installation	60

RETURN LETTER

Swopping Online System (SOS)
Title: User's Guide/Reference Manual/
Operating Guide/Installation Guide

RCSL No.: 31-D662

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

 **REGNECENTRALEN**
af 1979

Information Department
Lautrupbjerg 1
DK-2750 Ballerup
Denmark

information	repl.	ident	FB 820302	page	1/1
	RC8000	RC4000		class	EXT
subj. Supplement to RCSL No 31-D662, Swopping Online System (SOS)					
<p>MIPS/TS SW8100/1 Release 3.0 introduces a changed reaction to the BREAK command.</p> <p>Use of the BREAK command causes a new FP to be loaded in the user's process - without starting the 'RUNNING' process in its BREAK-routine.</p> <p>In installations where this changed reaction is unacceptable, one may alter the system trimming parameter "CONDITIONS" to preserve the former mode of reaction.</p> <p>For further information please refer to: RCSL No 31-D662, Swopping Online System (SOS), section 5.2.</p>					