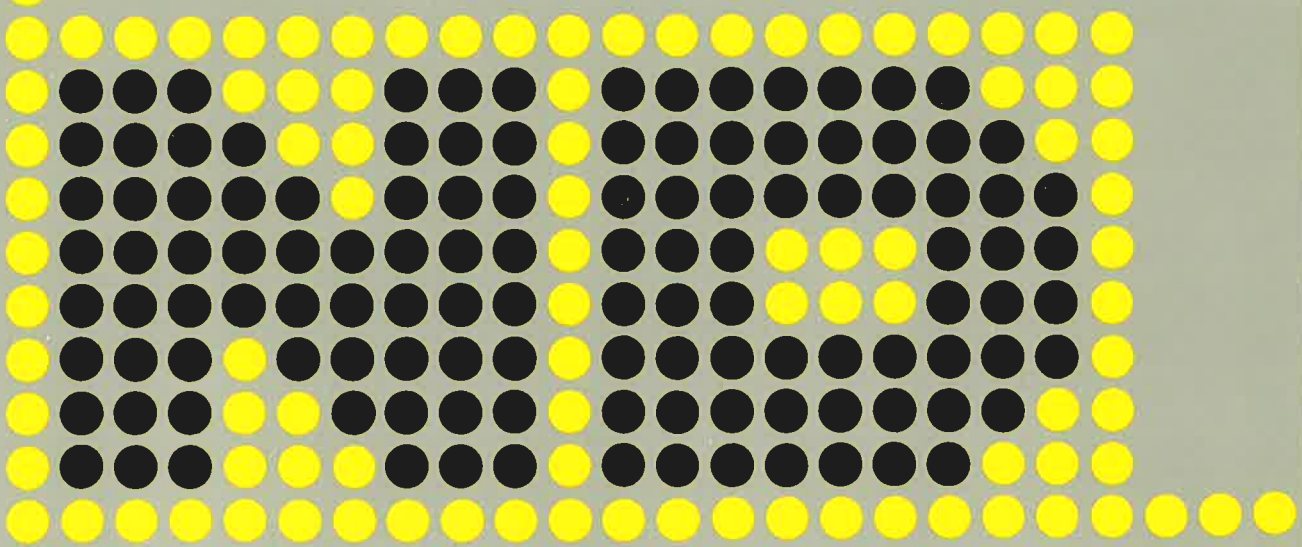
Norsk Data

SINTRAN III Tuning Guide

ND-30.049.1 EN



SINTRAN III Tuning Guide

ND-30.049.1 EN

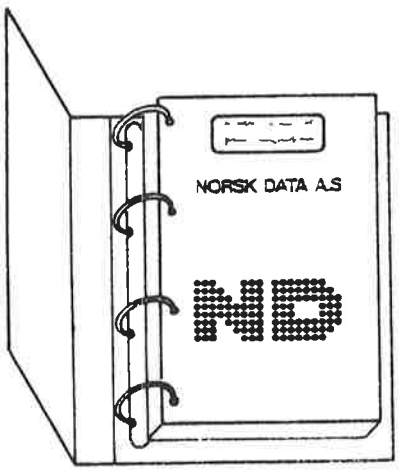
NOTICE

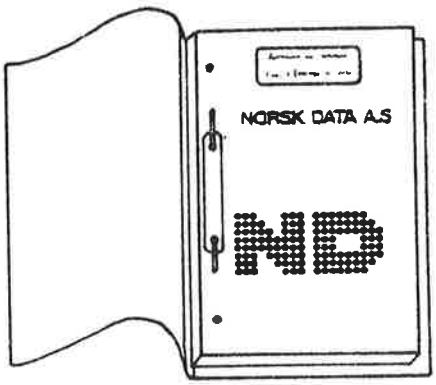
The information in this document is subject to change without notice. Norsk Data A.S assumes no responsibility for any errors that may appear in this document. Norsk Data A.S assumes no responsibility for the use or reliability of its software on equipment that is not furnished or supported by Norsk Data A.S.

This manual is in loose-leaf form for ease of updating. Old pages may be removed and new pages easily inserted if the manual is revised.

The loose-leaf form also allows you to place the manual in a ring binder (A) for greater protection and convenience of use. Ring binders with 4 rings corresponding to the holes in the manual may be ordered in two widths, 30 mm and 40 mm. Use the order form below.

The manual may also be placed in a plastic cover (B). This cover is more suitable for manuals of less than 100 pages than for large manuals. Plastic covers may also be ordered below.





A: Ring Binder

B: Plastic Cover

Please send your order to the local ND office or (in Norway) to:

Norsk Data A.S
Graphic Center
P.O. Box 25, Bogerud
0621 Oslo 6, Norway

ORDER FORM

I would like to order

..... Ring Binders, 30 mm, at nkr 20,- per binder

..... Ring Binders, 40 mm, at nkr 25,- per binder

..... Plastic Covers at nkr 10,- per cover

Name .....

Company .....

Address .....

.....

City .....

PRINTING RECORD

Printing	Notes
05/85	VERSION 01

SINTRAN III Tuning Guide
Publ.No. ND–30.049.1 EN

Norsk Data

Norsk Data A.S
Graphic Center
P.O.Box 25, Bogerud
0621 Oslo 6, Norway

Manuals can be updated in two ways, new versions and revisions. New versions consist of a complete new manual which replaces the old manual. New versions incorporate all revisions since the previous version. Revisions consist of one or more single pages to be merged into the manual by the user, each revised page being listed on the new printing record sent out with the revision. The old printing record should be replaced by the new one.

New versions and revisions are announced in the Customer Support Information (CSI) and can be ordered as described below.

The reader's comments form at the back of this manual can be used both to report errors in the manual and to give an evaluation of the manual. Both detailed and general comments are welcome.

These forms and comments should be sent to:

Documentation Department
Norsk Data A.S
P.O. Box 25, Bogerud
0621 Oslo 6, Norway

Requests for documentation should be sent to the local ND office or (in Norway) to:

Graphic Center
Norsk Data A.S
P.O. Box 25, Bogerud
0621 Oslo 6, Norway

Preface:

THE PRODUCT

The Tuning Guide deals with various tuning and performance aspects of ND systems.

THE READERS

The manual will be of interest to System Supervisors, (system) programmers and ND users with a general interest in the area of performance and tuning.

PREREQUISITE KNOWLEDGE

No specific prerequisite knowledge is required, but a general acquaintance with ND systems will be a great advantage.

THE MANUAL

The first chapters are mainly for personnel responsible for getting the best out of their systems. Measurement methods, interpretation of results and possible actions are covered in detail.

The remaining chapters give practical hints to system programmers on how to make their programs perform more efficiently.

RELATED MANUALS

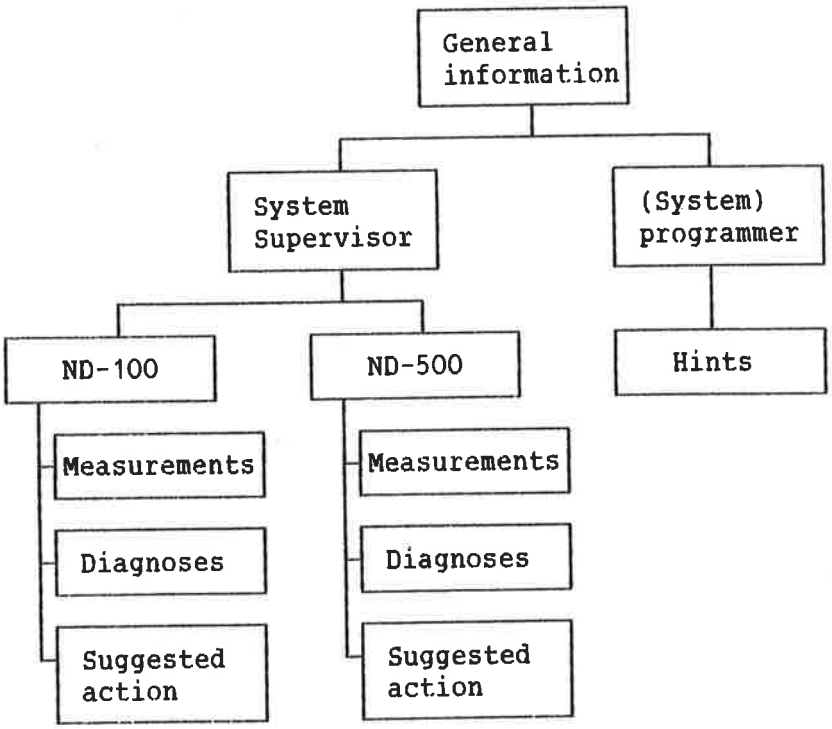
ND-500 Loader/Monitor	ND-60.136
SINTRAN III Reference Manual	ND-60.128
SINTRAN III Real Time Guide	ND-60.133
SINTRAN III System Supervisor	ND-30.003

TABLE OF CONTENTS

Section Page

1 INTRODUCTION

This figure shows how the manual is structured. ND-100 and ND-500 systems are discussed separately.



2 SOME BASIC PERFORMANCE ASPECTS

The two performance measures of greatest interest to us are RESPONSE TIME and CAPACITY. From the users' point of view, response time is regarded as the most relevant performance measure especially as regards the time sharing and transaction processing ones. However, from a system configuration point of view, capacity is a more natural performance measure.

A computer system can be regarded as a system of shared resources with queues, a so-called queueing system. A job submitted to the system will circulate between the various resources until it has been rendered. Resources may be physical resources, such as CPUs, disks and communication lines, or logical resources such as SIBAS or background programs for remote terminal access (TADs).

A job may have to wait in a queue for every requested resource. A long queue for a resource usually means that the resource is working at almost maximum capacity. So the underlying problem of a poor response time is often that one or several resources in the computer system have insufficient processing capacity. Such resources are referred to as system bottlenecks.

Thus, too long response times show the presence of bottlenecks with long queues. In order to improve performance, such bottlenecks must be removed. Basically, this is accomplished in two different ways:

1. The processing capacity must be increased at the critical resources, in general by adding more hardware.
2. The load on these resources must be reduced, by tuning the system or by changing the external load pattern (batch jobs delayed until the evening etc).

If response times are critical, one should not configure a system in which a shared resource is utilized close to 100% of its capacity. When this limit is approached, response times will usually skyrocket. As a rough rule (although this often represents worst case), the response time from a shared resource will be about twice, three times and ten times the average service time when the resource is utilized at 50%, 67% and 90% of its capacity, respectively (see fig. 1).

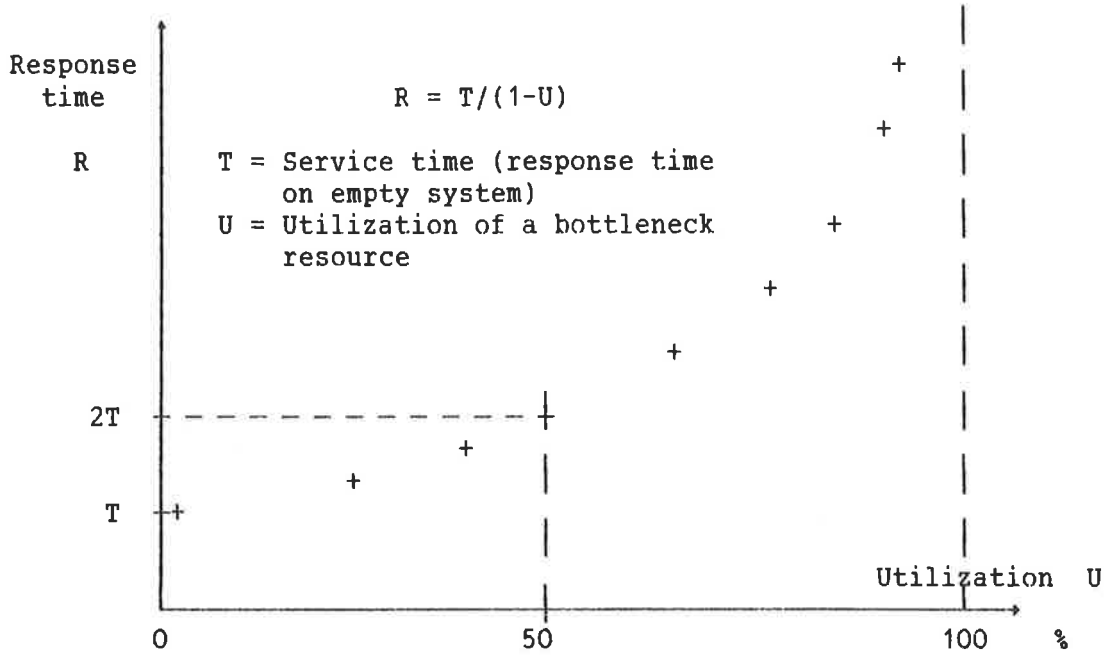


Fig. 1. Response time versus utilization of a shared resource

A response time of two to three times the service time, or two to three times the sum of the service time if several resources are involved each of which may be visited more than once, is usually acceptable. However, one should keep in mind that peak hour resource utilizations may be considerably higher than the average over the workday. In general, if fast responses are critical, you might say:

> Applied average load should not exceed about 65% of maximum system capacity.

In fact, unless estimates are quite conservative, one should plan for even lower utilizations.

On the other hand, the importance of good response times depends a good deal on the type of work involved - or job class. In many systems one strives for a happy coexistence of different job classes with different demands on response times.

For instance, many users do not mind too much if a batch job takes one or two hours instead of half an hour, provided that on-line work can go on simultaneously with little or no degradation of response times.

Proper handling of these matters is not only a capacity problem, but also a question of how the operating system

handles jobs of different importance (scheduling). To a certain extent, this can be controlled by tuning a number of system parameters.

3 MEASUREMENT TOOLS

A number of measuring facilities is present in SINTRAN III and ND-500-MONITOR. Each of them is described in detail in this section. "Standard" parameter values are suggested and hints given for the effective use of the tools.

3.1 ND-100

For ND-100 the following SINTRAN III measurement tools are available in a standard system:

- RT-PROGRAM-LOG
- PROGRAM-LOG
- HISTOGRAM
- SYSTEM-HISTOGRAM
- TIME-USED
- SYSTEM-HISTOGRAM and PROGRAM-LOG in combination

In addition, the SINTRAN-SERVICE-PROGRAM can be used for:

- Moncall-log (monitor call log)
- Swapping-log
- CPU-log
- Disk-access-log

These last four tools were included in the J-version of SINTRAN III. The CPU log facility is standard in all systems, but the other three have to be ordered when SINTRAN III is generated.

A significant difference between the traditional measurement tools and the newer ones in the SINTRAN-SERVICE-PROGRAM, is that the former are based on interrupt-driven sampling, while the latter are based on event tracing.

In the case of SAMPLING, various system parameters are read each time a sample is taken. The sources used for interrupts are the system hardware clock and various character devices. EVENT-TRACING, on the other hand, registers certain events when they occur, and must therefore be contained in the code used for generating or handling these events.

For example, the CPU log is based on counting the number of loops executed in the dummy program during a specific time interval, thus counting the number of occurrences of the event "CPU is idle", and comparing this with the corresponding number on an empty machine. The method used by tools of the sampling type to measure CPU load is to read the "previous level" register in the system. This registers the hardware level active immediately before the level from which sampling takes place. If the "previous level" was different from zero (idle level, see the section TIME-USED on page 13) then the sample registers the CPU as busy.

3.1.1 RT-PROGRAM-LOG

The RT-PROGRAM-LOG measures resource usage for a particular Real-Time program and/or the system as a whole. It is also possible to measure the resource demands submitted from any background terminal or batch processor, as each of them is internally connected to a Real-Time program (BAKxx, BCHxx or TADxx).

3.1.1.1 PARAMETERS FOR RT-PROGRAM-LOG

In this example SIBAS (RT NAME: SIBA) is measured. If you are interested only in measuring the overall system utilization, then just give ↵ instead of the name of an RT-program.

```
@RT-PROGRAM-LOG
RT NAME: SIBA
INTERVAL(SEC): 10
INTERRUPTS/SAMPLE: 1
LOG. UNIT NO.: 1207
INPUT/OUTPUT (0 OR 1):
LOG. UNIT NO.:
OUTPUT FILE:
```

INTERVAL:
The time during which results of the sampling will be accumulated before writing a report line on the log device. Use a short interval (5-10 seconds) if you are interested in rapid fluctuations. If the overall load is of interest, use a longer time (30-120 seconds). Default value (give ↵) is 60 seconds.

INTERRUPTS/SAMPLE:
The log device is the "clock" for the RT-PROGRAM-LOG. Non-printing characters are output to the terminal. This is why the cursor will disappear on some screen terminals, and also why it is impossible to use this function from a remote

terminal. Every "Nth" character a sample is taken of the current system state. "N" is the number of interrupts per sample. This parameter has a default value of 8.

Example:
The number of character interrupts per second from a 9600 baud terminal is given by 9600/11 = 873. Division by 11 is done because it takes 11 bits to represent one character in the asynchronous protocol. Likewise, the number of interrupts per second from a 600 baud terminal (a KSR Hardcopy system console, for instance) is given by 600/11 = 55.

Example:
Say we choose an INTERVAL of 10 seconds. We then want the number of samples taken during each 10 second period to be large enough to produce statistically significant results. If we use 1 interrupt/sample for a 600 baud terminal, we would be sampling at a rate of 55 samples/second. A 10 second period gives 550 samples, which is large enough to give us reliable results. If we were using 4 interrupts/sample, the number of samples in the 10 second period would be 550/4 = 137. This number is on the edge of being too small to produce reliable results. The calculation procedure is the same for all other line-speeds, only the numbers are different.

LOG. UNIT NO.:
This parameter may be used to identify one or two I/O devices for which the degree of utilization is to be logged. If a device is reserved, it is defined as being in use.

Example:
The RT-PROGRAM-LOG by default only logs activity on the first disk controller in the system. If your system has two controllers, logical unit 1207 (octal) has to be given to look at the use of this second controller. A list of logical device numbers in a system can be found near the beginning of the SINTRAN III listing for the configuration dependent part (Part II).

3.1.1.2 OUTPUT FROM RT-PROGRAM-LOG

CPU	SWAP	FILES	DISK	PASSIVE	IO-WAIT	UNIT 1207
25/38	02/05	00/05	10	00	75	00/32

The first figure of each pair for CPU, SWAP, FILES and the two logical units, shows the named RT-program's utilization in percentage of the corresponding resource. The figures for PASSIVE and IO-WAIT also refer to the named RT-program. The second figure in each pair and the DISK figure show the

total utilization in percentage of that resource.

CPU:
In the above output, the RT-program used 25% of the CPU, the total use of the CPU being 38%.

SWAP:
Out of 100 samples, the given program was performing paging 2 times and some other program was paging 5 times.

FILES:
This means "normal" use of files (user I/O).

DISK:
The approximate sum of the total SWAP and FILES-figures, as these two are the only sources for the disk-traffic.

These figures arise as follows: At each sample, it is checked whether the datafields for the DMA controller and the ND-100 swapper are free or reserved. How much the DMA controller and the ND-100 swapper are used, is reflected in the DISK and SWAP figures respectively. If the DMA datafield is reserved, while the swapping datafield is free, the sample is registered under FILES. Sometimes, the sum SWAP + FILES is slightly higher than the DISK figure. Apart from rounding errors, this may be due to the delay from reservation of the swapping datafield to reservation of the DMA datafield. Normally, this delay is very short, but it may be significant if there is heavy activity on high interrupt levels. A consequence of this mechanism is that swap-I/O from ND-500 will be registered under FILES, since it does not involve the swapping system in the ND-100-part of the system.

All percentages are related to the INTERVAL-time given.

Example:;
If the INTERVAL is set to 10 seconds, and the DISK-figure becomes 40%, then the disk-system (disk-datafield) has been occupied 4 seconds out of 10. This includes time to set up transfers, seek-time and transfer-time.

The CPU, PASSIVE, and IO-WAIT-figures for a specific RT-program, add up to a certain percentage figure. (In the above example they are 25 + 0 + 75 = 100%). This percentage will be equal or close to 100% if the load on the system is small, but starts dropping below 100% when the load increases. The missing percentages give us an idea of the queue-lengths in the system. When an RT-program is waiting for the CPU (or is in the queue waiting to reserve some I/O-device), it will not be classified into any of the groups defined by RT-PROGRAM-LOG. If a sample is taken when the RT-program is in the waiting queue of some device, this will be a "missing" sample.

3.1.1.3 SOURCES OF ERROR

The overhead from a 600 baud or about 55 characters/second terminal at 1 interrupt per sample is negligible (about 1% of the CPU). A 9600 baud terminal sampling at maximum rate (873 samples/second) will use approx. 17% of the CPU on an ND-100/CX.

The following table states the CPU-overhead (in%) on ND-100/CX as a function of line speed and the number of interrupts/sample (For ND-100 standard the overhead will be appoximately 25% larger):

Line speed	Interrupts/sample						
	1	2	4	8	16	32	64
300	1	0	0	0	0	0	0
600	1	1	1	1	1	1	1
1200	2	2	2	2	2	2	2
2400	4	4	4	3	3	3	3
4800	9	8	8	7	7	7	7
9600	17	16	15	15	14	14	14
19200	35	32	30	29	28	28	28

PROGRAM-LOG and the combination of PROGRAM-LOG and SYSTEM-HISTOGRAM are based on the same sampling method as RT-PROGRAM-LOG, and the overhead is nearly the same.

An easy-to-remember way of selecting the INTERRUPTS/SAMPLE-parameter is given in the table below. Divide the baud rate of the terminal being used by 300, and use this result as the number of interrupts per sample. This means that the sampling rate will be about 27 samples per second:

For a 300 baud terminal, use 1 interrupts/sample (1 % CPU-overhead)
For a 600 baud terminal, use 2 interrupts/sample (1 % CPU-overhead)
For a 1200 baud terminal, use 4 interrupts/sample (2 % CPU-overhead)
For a 2400 baud terminal, use 8 interrupts/sample (3 % CPU-overhead)
For a 4800 baud terminal, use 16 interrupts/sample (7 % CPU-overhead)
For a 9600 baud terminal, use 32 interrupts/sample (14 % CPU-overhead)

The table on CPU-overhead shows that the usage of higher sampling rates gives more accurate measurements, without increasing the overhead significantly.

The accuracy of the RT-PROGRAM-LOG is limited by the fact that the sampling itself runs on hardware level 10, and

therefore is not able to register activity on level 10 and higher correctly. Thus, for most "normal" workloads the results are quite reliable. However, in cases where the activity on hardware level 10 and upwards is a large part of the total CPU-consumption, the results are less reliable, and can only be used as an indication. (See the paragraph "Combination of SYSTEM-HISTOGRAM and PROGRAM-LOG" if you want to find out about the distribution of CPU-load over the various hardware levels.)

If there is a heavy load on the system, the report lines (which should be printed every INTERVAL) may be delayed because of the normal time slicing system,which is also used for the logging device. To correct this, you use the command "REMOVE-FROM-TIME-SLICE" in the SINTRAN-SERVICE-PROGRAM and then @PRIOR BAKxx <high priority, e.g. 100B> on the BAKxx of the logging device

A rather strange error occurring once in a while for the RT-PROGAM-LOG, is the following:
When the system is idle, the RT-PROGRAM-LOG may still show a rather high CPU-usage. For instance, a figure of 17 percent CPU has frequently been observed on idle systems with the log running from a 300 baud console, with
INTERRUPTS/SAMPLE = 1.

The explanation is:
Since each character contains 11 bits, the time between successive samples from a 300 baud device will be 11/300 seconds. The time between successive interrupts from the system clock is 1/50 second. Let us assume that an interrupt from the logging device happens to arrive while the software clock on level 13 is active. The former interrupt, which is to level 10, will be queued, and as soon as the activity on level 13 stops, sampling will take place and register "previous level" as non-zero, i.e. the CPU was busy. If this takes place at time zero, then 6 samples (i.e.
6 x (11/300) = 11/50 = 11 "clock ticks") later, the two interrupts "collide" again and the sample shows "CPU busy" once more. Thus, over a time period of length INTERVAL, 1/6 (=17%) of the samples indicates a busy CPU. This phenomenon is called phasing.

This occurs every time one out of 6 successive samples hits a clock interrupt, i.e. 6 out of 100 times. The best way to avoid this is to escape the log and start over. If possible, the log should run on an empty system until the first report line is displayed, in order to check whether phasing occurs or not.

3.1.2 PROGRAM-LOG

The PROGRAM-LOG measures the relative amount of CPU time used by each RT-program during a given time interval. The log is started by the command START-PROGRAM-LOG, which has one parameter, INTERRUPTS/SAMPLE. This is the same as the parameter used in the RT-PROGRAM-LOG command described on page 5. To stop the log, STOP-PROGRAM-LOG must be given together with the name of the output file for the results.

Example:

```
@START-PROGRAM-LOG
INTERRUPTS/SAMPLE: 30

@STOP-PROGRAM-LOG
OUTPUT FILE: TERMINAL

             PERCENT  SAMPLES
DUMMY          82       997
STSIN          00       0
RTERR          00       0
RTSLI          01       11
  .            .        .
BAK08          17       205
  .            .        .
```

In other words, during the time the log was operating, the program BAK08 (representing one of the terminals in the system) used the CPU in 17% and the DUMMY in 82% of the samples.

DUMMY is the current RT-program when no other program is ready for for execution (the system is idle).
It has priority 0.

3.1.3 HISTOGRAM

The CPU HISTOGRAM measures the amount of CPU time spent in different parts of the logical address space of an RT-program. The DEFINE-HISTOGRAM command defines the parameters for the histogram. They are: The name of the RT-program, the start address of the logical area to be logged, and the address interval. 64 equally sized intervals are logged. This means that the parameters you choose must satisfy the following equation (all figures and parameters are octal):

START-ADDRESS + INTERVAL * 100 < 200000

The START-HISTOGRAM command and the STOP-HISTOGRAM may start and stop the log. The results is printed by the PRINT-HISTOGRAM, which has one parameter, the name of the output file.

Example:

```
@DEFINE-HISTOGRAM
RT NAME: GARP
START-ADDRESS: 0
INTERVAL: 2000B

@START-HISTOGRAM

@STOP-HISTOGRAM

@PRINT-HISTOGRAM
OUTPUT-FILE: TERMINAL

                    PERCENT    SAMPLES

OUTSIDE               0          0 OUT OF 12571
SYSTEM:               1          126
     0-  1777         0          0
  2000-  3777         7          880
  4000-  5777         9          1131
     .                .          .
176000-177777         0          0
```

"SYSTEM" here means CPU-time spent on hardware level 1, protection ring 1, 2 or 3, i.e. monitor call code is executed on behalf of the specified RT-program. Sampling for the histogram is done by the system clock at each "tick". These occur at 20 ms intervals. Therefore, statistically, each sample represents 20 ms CPU time. So by multiplying a number of samples by 20 ms, you get the corresponding amount of CPU time.

3.1.4 SYSTEM-HISTOGRAM

The SYSTEM-HISTOGRAM command measures the relative (and absolute) amount of CPU time used within sections of the physical memory on a specified interrupt level. The interrupt level, the start address, and the size of the increments of the memory area to be logged, must be defined by the command DEFINE-SYSTEM-HISTOGRAM. 64 equally sized intervals are logged. To start and stop this log you use the

START-HISTOGRAM and STOP-HISTOGRAM commands. The results is be printed by PRINT-HISTOGRAM, which has one parameter, the name of the output file.

Example:
If you are interested in the CPU use on level 1 and in the address area (physical) 2000B to 3000B, then use the following command sequence:

```
@DEFINE-SYSTEM-HISTOGRAM
LEVEL: 1
START-ADDRESS: 2000
INTERVAL: 10

@START-HISTOGRAM

@STOP-HISTOGRAM

@PRINT-HISTOGRAM
OUTPUT FILE: TERMINAL


                        PERCENT    SAMPLES
OUTSIDE:                  03         127 OUT OF 3868
2000-2007                 00         0
2010-2017                 00         0
 .   .                     .          .
2740-2747                 97         3740
 .   .                     .          .
```

3.1.5 TIME-USED

The TIME-USED command (or monitor call 114 - TUSED) returns the sum of the CPU time used in inbyte/outbyte-routines (hardware level 4) and in the user program itself (hardware level 1, protection ring 0) since the terminal logged on (or since a batch job started).

It is important to notice that this is never the total CPU time used, since time spent on levels 14, 13, 12, 11, 10, 5, 3 and 1 (ring 1, 2 or 3) is not included. To cut it short, we can say that the operating system overhead (apart from monitor calls for certain character handlings) is not included in TIME-USED.
Here is a list to give an idea of the activities on the 16 different hardware levels:

Level	Activity
0	The system is in the idle loop (DUMMY is current RT-program)
1	All RT-programs execute on this level
2	Not used
3	The kernel of SINTRAN III executes here
4	Inbyte/outbyte monitor calls
5	XMSG
6	Not used
7	Not used
8	Not used
9	Not used
10	Driver routines for character device output
11	Driver routines for mass storage devices
12	Driver routines for character device input + ND-500 driver + HDLC output driver
13	Updating of real-time clock + HDLC input driver
14	Internal interrupts (page faults, power fail, ..)
15	Not used

The fraction of CPU time not included in TIME-USED varies from program to program, and is often in the area of 1-50% of total CPU time used. If we exclude certain monitor calls for handling characters, we might say that the more monitor calls a program is executing, the greater is the fraction of CPU-time not included in TIME-USED.

3.1.6 COMBINATION OF SYSTEM-HISTOGRAM AND PROGRAM-LOG

This combination will give some extra information. The following command sequence could be used:

```
@DEFINE-SYSTEM-HISTOGRAM
LEVEL: 1
START-ADDRESS: 2000
INTERVAL: 10

@START-PROGRAM-LOG
INTERRUPTS/SAMPLE: 30

@STOP-PROGRAM-LOG
OUTPUT FILE: TERMINAL
```

Example:
The output could look like this:

```
        PERCENT  SAMPLES
 DUMMY:  95  1252
 STSIN:  00  0
   .      .  .
   .      .  .
 BAK14:  00  0
 BAK15:  02  29
 BAK16:  00  0
   .      .  .
   .      .  .
```

← This is as before (PROGRAM-LOG)

```
     PERCENT SAMPLES
LEVEL 0 : 93  1225
LEVEL 1 : 03  44
LEVEL 2 : 00  0
LEVEL 3 : 01  19
LEVEL 4 : 00  5
LEVEL 5 : 00  6
LEVEL 6 : 00  0
LEVEL 7 : 00  0
LEVEL 8 : 00  0
LEVEL 9 : 00  0
OTHER LEVELS: 01 19
```

← This is new

```
                PERCENT SAMPLES
OUTSIDE:            100  44 OUT OF 44
   2000-   2007:    00  0
     .       .       .  .
     .       .       .  .
   2770-   2777:    00  0
```

This is as before (SYSTEM-HISTOGRAM), except that sampling is done by the terminal output driver rather than the system clock

The extra information we get here is a picture of how the CPU load is distributed over the various hardware levels. In this example the system has been on level 0 (idle) 93% of the time, on level 1 (user RT-programs) 3% of the time and on level 3 (operating system) 1% of the time. Other levels (10, 11, 12, 13, 14) account for 1% of the time the log has been running. We also see that there has been some activity on level 4 (byte I/O) and level 5 (XMSG).

It may seem a little strange that the number of samples registered for the RT-program DUMMY is higher than the number on level 0 (1252 versus 1225). The reason for this is that the system has at all times one RT-PROGRAM which is defined as "current", and which is the basis for the PROGRAM-LOG part of the measurement. An RT-program may well be "current" though the system is handling information completely irrelevant to that particular program, particularly various interrupt handlings. That is why the number of samples under DUMMY is generally a few percent higher than the number of samples on level 0.

3.1.7 TOOLS IN SINTRAN-SERVICE-PROGRAM

These tools were implemented in the J-version of SINTRAN III.

3.1.7.1 MONITOR CALL LOG (MONCALL-LOG)

By means of this function you can count the number of monitor calls executed in the system as such, or by one specific program.

Example:;

```
@SIN-SER
*MONCALL-LOG
FUNCTION: START-MONCALL-LOG
LOG MONCALLS FOR ONLY ONE PROGRAM (DEFAULT IS YES)? NO
FUNCTION: STOP-MONCALL-LOG
FUNCTION: PRINT-MONCALL-LOG
OUTPUT FILE: MONCALLS
FUNCTION: EXIT
*EXIT
```

The MONCALLS file might look like this:

```
MONCALL NUMBER  NUMBER OF TIMES USED

      0:              0
      1:             19
      2:             32
      3:              0
      4:              0
      5:              0
      6:              0
      7:              0
     10:              0
      .               .
      .               .
    374:              0
    375:              0
    376:              0
    377:              0
```

This output should require no further explanation.

Use the HELP command (after FUNCTION:) to find out which commands are available in MONCALL-LOG.

For programmers who want to use as little CPU time (and I/O-time) as possible, the basic rule is to write programs that execute the smallest possible number of monitor calls.

3.1.7.2 SWAPPING-LOG

This command is useful for getting a picture of how many disk accesses are generated because of swapping (in the total system, or by one specific program).

Example:
We could use the following command sequence:

```
@SINTRAN-SERVICE-PROGRAM
*SWAPPING-LOG
FUNCTION: START-SWAPPING-LOG
LOG SWAPPING FOR A SPECIFIC PROGRAM (DEFAULT IS YES)? NO
FUNCTION: READ-SWAPPING-LOG
```

```
  TOTAL NUMBER OF PAGE FAULTS WITHOUT DISK ACCESS          101948
  TOTAL NUMBER OF PAGE FAULTS IN RT-COMMON                      0
  TOTAL NUMBER OF PAGE FAULTS ON LEVEL 4                        3
  TOTAL NUMBER OF PAGE FAULTS ON LEVEL 1                     7515
  TOTAL NUMBER OF PAGES SWAPPED OUT (WRITTEN TO DISK)        3230

FUNCTION: EXIT
*EXIT
```

Explanation of the previous table:

1. PAGE FAULTS WITHOUT DISK ACCESS

Gives the number of times there has been a page fault due to the "window" mechanism in SINTRAN III. A window page is used for example to access data in the terminal datafields outside the first four banks in SINTRAN III. This always generates an internal page fault (which means that a certain amount of CPU time is spent handling the fault), but never disk access.

2. PAGE FAULTS IN RT-COMMON

This is the same type of page faults as in 1, but in the RT-COMMON instead of in the window. No disk access is generated.

3. PAGE FAULTS ON LEVEL 4

These are page faults generated by the SINTRAN III byte-I/O routines running on hardware level 4. In most cases no disk access is generated. The uncertainty about whether a page fault like this leads to a physical disk access or not, should cause no worry, because these page faults are normally very few.

4. PAGE FAULTS ON LEVEL 1

This is the normal type of page faults which causes access to the segment files on disk. Each fault generates one disk access if no page has to be written back, and two disk accesses if a used page (WRITTEN-IN-PAGE-bit set) has to be written back before the new page can take its place.

5. PAGES SWAPPED OUT (WRITTEN TO DISK)

This is the case where a page has to be written back to disk (swapped out, WIP-bit set) before a new page can be swapped in. In our example this occurred 3230 times out of 7515 possible times.

The total number of physical disk accesses in this example may therefore be calculated as:

```
7515 + 3230 = 10745
====================
```

The command READ-SWAPPING-LOG can be executed at any time to find out how swapping accesses accumulate.
The command SWAPPING-LOG should be used if you would like to have a report line printed at fixed intervals (use the the INTERVAL command).

Use the HELP command (after FUNCTION:) to find out the available commands in in SWAPPING-LOG.

3.1.7.3 CPU-LOG

The command CPU-LOG prints the CPU-utilization as a percentage number at specified intervals.

Example:

```
@SINTRAN-SERVICE-PROGRAM
*CPU-LOG
INTERVAL IN SECONDS (DEFAULT IS 30 SECS): 60
OUTPUT FILE:

 10
 12
 13
 12
```

We see that the average CPU-load is around 12%.

The CPU load calculation is based on the number of times the CPU runs through an idle loop when the system is completely idle. This number is stored in location 342 in resident (J-version), and the default value is approximately (1750B) for ND-100/CX CPUs. It must (may) be calibrated on each individual system (when idle) by the command FIND-CPULOOPTIME. After executing this command (it uses 30 seconds to count), the number of CPU loops per second (decimal) is printed on the terminal. The variable CPULOOPTIME may then be changed to this value by CHANGE-VARIABLE, a SINTRAN-SERVICE-COMMAND, to make the CPU-LOG as accurate as possible for your system.

3.1.7.4 DISC-ACCESS-LOG

This command is used when you want a count of disk accesses. Various conditions may be specified, such as controller to log, which unit, read or write. In addition, if a certain part of a disk is particularly interesting, it may be logged.

The results may be displayed directly on your screen, as in the following example, or they may be written to a disk file. This file must be contiguous and of type :LOG. The results will be stored in binary format. You must either write a special program to read the file, or you may find the disk address of the file in location 36-37B in the file's object entry and use the command @DUMP-PAGE on the pages of interest.

Let us look at a simple example where we count all disk accesses in the system:

```
@SINTRAN-SERVICE-PROGRAM
*DISC-ACCESS-LOG
FUNCTION: START-DISC-ACCESS-COUNTER
COUNT ALL DISC ACCESSES (DEFAULT IS YES)? YES
FUNCTION: LOG-DISC-ACCESS-COUNTER
INTERVAL IN SECONDS (DEFAULT IS 60 SECS): 30
```

TOTAL DISK ACCESSES IN INTV./ACCUMULATED		WRITE ACCESSES IN INTV./ACCUMULATED		READ ACCESSES IN INTV./ACCUMULATED	
5/	5	2/	2	3/	3
8/	13	3/	5	5/	8
15/	28	5/	10	10/	18

In addition, special error information may be read from the disk driver. Use the HELP command (after FUNCTION:) to list the commands available under DISC-ACCESS-LOG.

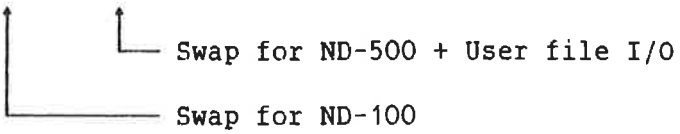
==========

3.2 ND-500

The front end part of any ND-500 system is a regular ND-100/CX processor, which means that the measurement tools for the front end CPU are the same as those described for ND-100. However, as regards a 500 system, you must to be aware of:

ND-100 and ND-500 have their own (independent) swap systems. So, when the 500 processor is swapping, RT-PROGRAM-LOG will not see this as swapping for ND-100, and therfore classify it as "FILES":

```
CPU    SWAP  FILES    DISK
25      02     08      10
```



All other measurement tools for ND-500 can be accessed from the ND-500 monitor. These tools are described in the "ND-500 Loader/Monitor" manual. They are:

- HISTOGRAM
- MONITORCALL-LOG
- PROCESS-LOG
- SWAPPING-LOG
- LIST-EXECUTION-QUEUE

The HISTOGRAM and PROCESS-LOG are based on interrupt-driven sampling by the system clock (the measurement runs entirely in the 100 CPU). The MONITORCALL-LOG and SWAPPING-LOG are based on event-tracing in the 500 monitor (which runs in the 100 CPU) and the 500 swapper (which runs in the 500 CPU), respectively.

Since the 500 processor does not have an interrupt system and no I/O-system of its own, and sampling for HISTOGRAM and PROCESS-LOG is done by drivers running in the 100-part, results are more correct than the corresponding tools for the 100-CPU. In particular, it is possible to measure accurately the resource usage by one process in the 500-part even though other processes are active simultaneously. Such measurements are often very inaccurate on a 100-system.

The formal use of these commands (parameters and so on) is shown in the Loader/Monitor manual, and is not repeated

here. In the following sections is some additional information.

3.2.1 HISTOGRAM

This command is used to find out where the 500 CPU time is spent inside the address area of one program (domain). Remember that the 32-bit addresses may be given a "short-form". The normal start-address 10000000004B may be specified as 1'4B.

Example:
The address area 2000B - 50000B on segment 1 is histogrammed.

```
@ND
ND-500 MONITOR  VERSION E    83.12.20 / 83.12.21
N500: SET-HISTOGRAM 1'2000B,1'50000B,,
N500: START-HISTOGRAM
N500: <DOMAIN-NAME>
   .          .
N500: OUTPUT-FILE HISTO
N500: PRINT-HISTOGRAM
N500: EXIT
```

The HISTO file could look like this:

```
HISTOGRAM  OF  <DOMAIN-NAME>
   PRINTED:  11. 9.36      29 NOVEMBER 1984

1000002000B:     0.0
1000002461B:     0.1
1000003142B:     0.0
1000003623B:     0.0
1000004304B:     2.7   ******
1000004765B:     8.6   ********************
1000005446B:     3.6   ********
1000006127B:     2.7   ******
1000006610B:     2.9   ******
1000007271B:     0.7   *
1000007752B:     0.0
1000010433B:     0.0
1000011114B:     0.3
1000011575B:     0.3
1000012256B:     0.1
1000012737B:     2.5   *****
1000013420B:     0.3
1000014101B:     7.5   ******************
     .            .
     .            .
1000033473B:     1.1   **
1000034154B:     0.1
1000034635B:     5.3   ************
1000035316B:     0.7   *
1000035777B:     0.1
1000036460B:     0.0
1000037141B:    21.4   ************************************************
1000037622B:     1.8   ****
1000040303B:     9.3   **********************
1000040764B:     0.1
1000041445B:     4.0   *********
1000042126B:     0.3
1000042607B:     0.0
1000043270B:     1.1   **
1000043751B:     0.3
1000044432B:     0.3
1000045113B:     0.3
1000045574B:     0.7   *
1000046255B:     2.7   ******
1000046736B:     0.0
1000047417B:     0.0
OUTSIDE:        16.1    ************************************
NO. OF SAMPLES:         1041B
```

We can easily see where the program's CPU time is spent, and it is common to "zoom" in on smaller address areas of particular interest (for example from 1'37141B to 1'37622B in this case).

Note that each sample represents 20 ms of CPU, so the total number of samples multiplied by 20 ms gives the total CPU-time used. It corresponds with the information obtained by using the ND-500-MONITOR command TIME-USED.

3.2.2 MONITOR CALL LOG

This command counts the number of monitor calls executed, and is very useful if the ND-100 part of a 500 system is overloaded. The command-sequence:

```
@ND-500-MONITOR
N500: START-MONCALL-LOG ALL
N500: <DOMAIN-NAME>
        .
        .
N500: OUTPUT-FILE "MON-CALLS"
N500: PRINT-MONCALL-LOG
N500: EXIT
```

will log all monitor calls executed in the system, and might produce the following MON-CALLS file:

```
      MONITOR CALL LOG  OF  <DOMAIN-NAME>
         PRINTED:  18.42. 2       13 NOVEMBER 1984

   0B -          1B          5B          0B          0B
                 0B          0B          0B          0B
  10B -          0B          0B          0B          3B
                 0B          0B          3B          3B
  20B -          0B          0B          0B          0B
                 0B          0B          0B          0B
  30B -          1B          0B          0B          0B
                 0B          0B          0B          0B
  40B -          0B          2B          0B          6B
                 0B          4B          0B          0B
  50B -         11B          0B          1B          0B
                 0B          0B          0B          0B
  60B -          0B          0B          3B          0B
                 0B          0B          0B          0B
  70B -          0B          1B          1B          0B
                 6B          0B          2B          0B
 100B -          0B          0B          0B          0B
                 0B          0B          0B          0B
 110B -          0B          0B          0B          5B
                 0B          0B          0B          5B
 120B -          0B          0B          0B          0B
                 0B          0B          0B          0B
 130B -          0B          0B          0B          0B
                 0B          0B          0B          0B
 140B -          0B          0B          0B          4B
                 6B          0B          0B          0B
 150B -          0B       4026B          0B          0B
   .             .           .           .           .
   .             .           .           .           .
 500B -          0B          0B       2614B         56B
              16330B          0B          0B          0B
   .             .           .           .           .
```

The left column contains the monitor call number. The other columns contain the number of executions of the various monitor calls. In this case we notice that most monitor calls have been executed a reasonably low number of times. A few calls have a very high number count (monitor call number 504 has been executed 16330B times). The next step would be to find out which program causes this, and if it really is necessary to use that many monitor calls.

3.2.3 PROCESS-LOG

3.2.3.1 PROCESS-LOG-ALL

This command gives the best overall picture of what is going on in a 500 system. The output looks something like this:

PROCESS LOG ALL 12.39. 3 29 NOVEMBER 1984

0	1	2	3	4	5	6	7	8	9	10	11	OTHERS	IDLE	SWAP
1	0	2	55	3	1	0	0	0	0	0	0	0	38	44
0	0	1	54	5	1	0	0	0	0	0	0	0	39	89
0	0	5	75	11	2	0	0	0	0	0	0	0	7	57
3	0	18	42	6	1	0	0	0	0	0	0	0	30	69

There are a number of factors we can observe from this table:

- All figures are in percent of the INTERVAL given (as for RT-PROGRAM-LOG).
- Process 0 is the 500 swapper itself, so we can get an idea of how much CPU resources it takes to administrate the swapping system (in this case only 0-3%).
- The total CPU load on the system has been 61-93% (100 minus IDLE-figure).
- Total swap rate has been 44 - 89%. The problem is that this SWAP figure does not distinguish between real swapping (lack of physical memory for code and its lokal data), and swapping caused by programs using the "file-as-segment" facility for data files, which is actually user-I/O.
- Process number 3 is using most of the CPU resources (42-75%), which most often implies that this user is running some heavy job like compiling or text formatting.
- Process number 5 is only using 1-2% of the CPU, and this is typical for a light edit session (PED, NOTIS-WP).
- Process 2 and 4 are medium heavy users.

- To find out what the users are really working with, use the WHO-IS-ON command in the ND-500-MONITOR, and then the TERMINAL-STATISTICS command in SINTRAN III. The PROCESS-STATUS command is also very useful to see how CPU time is distributed among the various 500 processes. To look more closely at one specific process, see next paragraph.

3.2.3.2 PROCESS-LOG-ONE

We select one 500 process of particular interest (number 15D in this example), and get the following output:

```
LOGGING OF PROCESS  15   14.13.10   29 NOVEMBER 1984

.........................IDLE
!   .....................WAITING FOR SWAPPER
!   !   .................USING SWAPPER
!   !   !   .............IN MONITOR CALL
!   !   !   !   .........ACTIVE
!   !   !   !   !   ....WAITING FOR CPU
0   0   0  48  48   2
4   3  16  32  24  18
0   0   1  36  55   5
0   0   0  49  46   4
0   1   2  50  39   6
```

From this we can tell:

- WAITING FOR CPU is low, so the ND-500 execution queue has been relatively short most of the time.

- ACTIVE shows that this program has been using 24-55% of the 500 CPU.

- IN MONITOR CALL gives an idea of how much time is spent on I/O. If this figure gets very high, we must expect that the program is not performing as good as one would like.
  There is one exception: A program waiting for input from the user will normally be logged as IN MONITOR CALL most of the time (close to 100%). This does not mean the program is inefficient, but only that the system spends most of its time waiting for the user to type something.

- USING SWAPPER tells us how this program is using the paging system.

- WAITING FOR SWAPPER means somebody else is using the swapper, and this program is in a queue to get access to it.

- IDLE means the program is passive.

3.2.4 SWAPPING-LOG

This log is meant mainly for testing purposes. It is not of general interest, and therefore not described here.

3.2.5 LIST-EXECUTION-QUEUE

This is a very useful command to get an idea of what priorities the different tasks are executing on. The output looks like this:

```
CURRENT EXECUTING PROCESS : NONE
     PROCESS NO     PRIORITY
           0B          300B  <-- ND-500 swapper
           1B           55B  ---+
           6B           55B     |  Very interactive users
           7B           55B     |- PED, NOTIS-WP, NOTIS-CALC
          10B           44B     |  User applications
          11B           44B  ---+
           3B           41B  ---+
           2B           41B     |- SIBAS Processes
           4B           40B  ---+
           5B           40B     |- SIBAS Servers
          12B           30B  ---+
          13B           24B     |- Heavy interactive jobs
          14B           20B     |  Compiling, formatting
          30B           20B  ---+
          15B           16B     |- Batch jobs
```

- See the chapter on Time Slicing in SINTRAN III on page 44 for a closer description of how priorities are assigned to different users.

- This execution list is sorted by priority, and the swapper (process 0) is the highest priority process.

- Very interactive users (typing break characters often) are mostly on priorities 55B and 44B.

- SIBAS and its servers are on priorities 41B and 40B.

These are RT processes in the ND-500 (can be verified by using the command WHO-IS-ON). They are not time sliced, as are the monitor calls (or commands), because SET-PRIORITY has been used. This means the priority is static, but can be changed (tuned) by the SET-PRIORITY command. This command has no meaning for time sliced processes.

- Heavy interactive jobs (or MODE jobs) are generally in the priority area 20B - 30B.

- Batch jobs spend most of their time on priorities 16B and 20B.

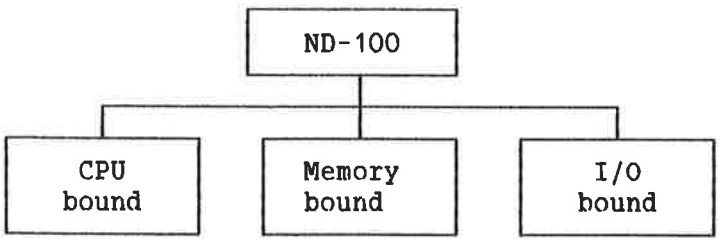
4 DIAGNOSIS

The most important question to answer in a system with performance problems is this:

What is the real bottleneck ?

The way to find out is to use the measurement tools described in this manual, and to interpret the results correctly. Let us look at some typical problem situations for ND-100 systems and ND-500 systems.
Note that I/O is used in the sense of mass storage I/O, as terminal I/O only contributes to the use of CPU resources.

4.1 ND-100



Measurements will have to lead us to one of the three conclusions in the above picture, unless they reveal some obvious explanation (like an RT program on high priority using most of the CPU, which makes it even simpler to diagnose).

4.1.1 CPU BOUND

This is the conclusion if RT-PROGRAM-LOG or CPU-LOG show a very high utilization of the CPU (70-100%) over long periods of time, while SWAP and FILES are at considerably lower levels. This situation is quite typical for ND-100 systems.

4.1.2 MEMORY BOUND

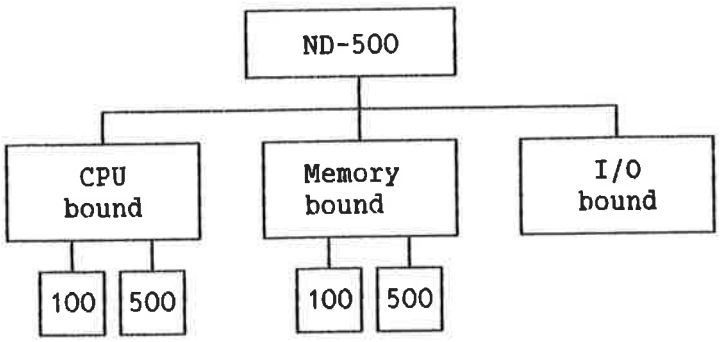
This is the situation where SWAP (from RT-PROGRAM-LOG) grows higher than 10-20% over long periods of time (several minutes). This is a problem in many of the installed ND-systems today, and has a very negative impact on response times.

4.1.3 I/O BOUND

Occurs if FILES (from RT-PROGRAM-LOG) grows higher than both CPU and SWAP for long periods of time. You will almost never see this in an ND-100 system.

==========

4.2 ND-500



We can use the same categorization for an ND-500 performance

problem as for an ND-100. However, due to the fact that any ND-500 system is a multi CPU system, the diagnoses are more complex, as illustrated above.

Furthermore, we will limit the diagnose to an ND-500 system where no user jobs are run in the ND-100 part of the system. We do this to simplify some of the arguments, and also because it is the way any ND-500 system should be run. The reason for this is the fact that the front end ND-100 has sufficient work to do acting as the ND-500 I/O processor. If more work is put on it, we must expect that either the users of ND-100 programs, or the users of ND-500 programs (or both !) will not be satisfied. The only tuning tool in a case like this (apart from allocating memory correctly) is the time slicing mechanism (to get the best out of a far from ideal situation).

4.2.1 CPU BOUND

We have two possible situations here. Either the front end ND-100/CX CPU is overloaded (use RT-PROGRAM-LOG or CPU-LOG), or the main ND-500 CPU is running at a very high average utilization (use PROCESS-LOG-ALL).

4.2.2 MEMORY BOUND

Again, either the front end CPU is running with a high SWAP rate, or the ND-500 CPU is swapping heavily (can be seen from PROCESS-LOG-ALL). The two CPUs have their own independent swapping systems, and both situations may reduce total system performance considerably. See the next section for further explanation.

Be aware of the fact that even if the system has sufficient memory in its hardware, it may not be optimally distributed (by software) between the two CPUs. See more about this in chapter Use of Memory in an ND-500 System, chapter 7.

4.2.3 I/O BOUND

This state may easily arise in an ND-500 system, as the main CPU is quite powerful. It will manifest itself in one of two

possible ways, depending on whether the "file-as-segment" facility is in heavy use or not:

1. File-as-segment not heavily used

The FILES column in the RT-PROGRAM-LOG shows the highest usage at the same time as no or little SWAP (from PROCESS-LOG-ALL) goes on in the ND-500.

2. File-as-segment in heavy use

The FILES column in the RT-PROGRAM-LOG shows the highest usage in the system, and the SWAP rate in the ND-500 is equally high (or higher). This situation occurs because the file-as-segment concept uses the swapping system instead of the normal I/O-system, and because the RT-PROGRAM-LOG classifies everything which is not ND-100 SWAP, as FILES.

So, this is the case where the SWAP for an ND-500 seems to indicate a memory bound system, but where it in fact reflects an I/O-bound system. A typical example could be a SIBAS backend system. Most of ND's system software use the file-as-segment I/O-method.

There is also one more trap to avoid: ND-500 programs doing I/O in a very inefficient manner (wrong monitor calls, too many monitor calls). The result is always a situation where the ND-100 front end CPU is overloaded, which then is just a symptom of the real problem.

5 SOLUTION

At this point, we have performed the necessary measurements, and we have been able to pin point the most critical factor of the system (the bottleneck resource). We now know that the computer system is running badly due to the scarcity of this resource. There are usually only two ways of solving this kind of problem:

1. Add more of the bottleneck resource to the system.
2. Use less of it.

The first option means buying more hardware. The second one involves finding out where the resource is being used up. That is, in which programs and where in these programs. After the program (area) has been isolated, the choice is between:

1. Optimizing the program with respect to the resource.
2. Lowering the relative priority of the program, accepting that it will run slower.
3. A mixture of alternatives 1 and 2 above.

We will now look more closely into the possible problem states, and again it seems natural to distinguish between ND-100 and ND-500 systems.

==========

5.1 ND-100

5.1.1 CPU BOUND

If the CPU can no longer be upgraded (hardware), and the programs running in the system are reasonably effective (use PROGRAM-LOG and HISTOGRAM), then this system has reached its capacity limit. Only one specific hardware upgrade may help: Substitute the non-buffered terminal interfaces with buffered ones. The effect of this upgradeing is described in section Buffered Terminal Interface, page 59.

Total capacity can, of course, be increased by adding another complete system (normally connected by COSMOS), or upgrading to some ND-500 system (if possible).

Note that a very high utilization of the CPU (70-100%) does not necessarily mean that the situation is disastrous. If the different tasks in the system have priorities according to their importance, this can - on the contrary - be an ideal situation (maximum throughput). However, if the very time critical tasks consume more than 60-70% of the CPU, we must expect the response-times for those tasks to be too long.

5.1.2 MEMORY BOUND

This problem is very easily cured by installing more memory (hardware). Without installing more memory, it can only be solved by running fewer tasks and/or users.

5.1.3 I/O BOUND

ND-100 systems with this problem (very few) can be hardware-upgraded with for instance extra I/O controllers. But this only helps if the program systems can use this extra hardware effectively (files spread over several drives, balanced use of files, etc.). This must be considered in each case (see chapter 9). In software, the most common improvement potential lies in increasing block sizes in I/O-transfers.

==========

5.2 ND-500

5.2.1 CPU BOUND

If the ND-500 CPU is heavily used, we are normally in a situation where the system is running all right, assuming that programs are fairly optimized and that priorities in the system is in order. In the hardware, an upgrade from the smallest ND-500 CPU to the biggest one is no problem (user/program transparent), and the AX option can be used for vector operations to get more out of the CPU.

If the ND-100 front end CPU is the bottleneck, we must find out why (see chapter on Diagnoses, I/O bound ND-500 system). If no software adaptions can be made, the system can be upgraded with buffered terminal interfaces.
See the description of this in the section Buffered Terminal Interface on page 59.

5.2.2 MEMORY BOUND

If total memory size is too small, more memory must be added. The allocation of memory done by software (see chapter 7) between the ND-100 and ND-500 processors should always be checked. The system should be tuned so that the front end ND-100 swaps no more than 5-10% of the time (run RT-PROGRAM-LOG). A higher swapping level effectively reduces the total ND-500 system performance.

Note that the physical size of local memory versus the size of multiport memory, is also a tuning point (explained in chapters 7 and 8).

5.2.3 I/O BOUND

If the file-as-segment access method is in extensive use, the situation can always be improved by adding more memory, as this reduces the overall swap rate. However, there is a limit to this. We can not expect to install sufficient memory to keep a large fraction of databases in memory (several hundred megabytes), but we should try to keep substantial portions of the index tables in memory, to make the system less I/O bound.

If file-as-segment is not very much used, the solution is the same as for an I/O bound ND-100 system. For more information, see the chapter Example of I/O-system Configuration, page 42.

6 I/O-CAPACITY

6.1 FUNCTIONAL OVERVIEW

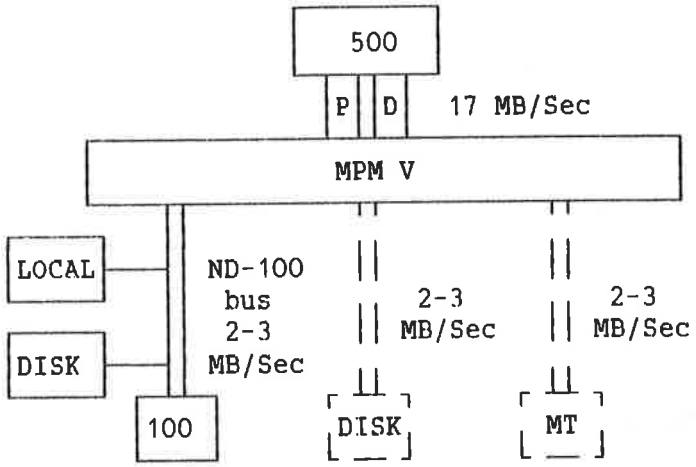


Figure 1. ND-500 system - functional overview.

The figure shows a standard ND-500 system, with optional controllers and channels stipled. The bandwidth over the two 32-bit ND-500 ports (one for program and one for data) is 17 MB/second in a standard system using MPM V. This bandwith may limit the speed of the CPU if cache hit is low, and if programs are very memory intensive.

The ND-100 bus has a capacity of 2 - 3 MB/second when accesses go into Multi Ported Memory, and close to 4 MB/second when accesses are to Local Memory.

Additional DMA-channels (stipled) also have a capacity of 2 - 3 MB/second each. A more accurate figure than 2 - 3 MB/second is difficult to state, as a certain amount of buffering takes place in the memory system on write accesses. The combination of read and write accesses in each individual program will decide the exact effective speed of the bus.

6.2 "TIME-SHARED" SYSTEM

By the expression "Time-shared" system, we mean a system where most users are doing work that generates disk accesses on a page by page basis.

Example:

A typical page fault in ND-500 is caused by the use of file-as-segment for file I/O. The column to the left shows the best case/fastest hardware, the column to the right the worst case/slowest hardware.

ONE DISK-ACCESS TRANSFERRING ONE PAGE

	Elapsed time	
500-CPU	3 - 4 MS	(ND-500 swapper)
100-CPU	4 - 5 MS	(Disk driver in ND-100)
SEEK	20 - 33 MS	(Average)
LATENCY	8 - 8 MS	(Average)
TRANSFER	1 - 3 MS	(Time for 2 KB)
Sum	36 - 53 MS	

——→ Max 19-28 ACCESSES/SEC
——→ TRANSFER RATE: 38-56 KB/SEC
——→ CHANNEL LOAD : 2-3%

This means that one disk drive, reserved (used) 100% of the time, can only account for a 2-3% average channel (bus) load when used in a page by page mode. Therefore, adding an extra DMA-channel for disk I/O will have a negliable effect in this situation.

6.3 "TAILORED APPLICATION" SYSTEM

This is a case where user applications take advantage of the possibilities of doing I/O in a very efficient way.

Example:

Transfers to contiguous files (can also be used for magnetic tape) are run with a physical transfer size of 16 pages (32 KB):

ONE DISK-ACCESS TRANSFERRING 16 PAGES

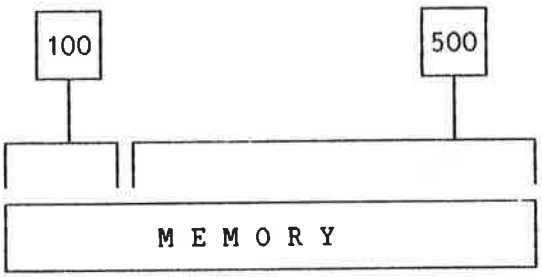
	Elapsed time	
500-CPU	0 - 1 MS	(ND-500 swapper)
100-CPU	4 - 5 MS	(Disk driver in ND-100)
SEEK	0 - 5 MS	(Usually same or neighbour)
LATENCY	8 - 8 MS	(Average)
TRANSFER	16 - 48 MS	(Time for 32 KB)
Sum	28 - 67 MS	

⟶ Max 15-36 ACCESSES/SEC
⟶ TRANSFER RATE: 0.5-1.2 MB/SEC
⟶ CHANNEL LOAD : 25-60%

This job now makes heavy use of the channel capacity (25-60%). If we want to run two or more of these jobs simultaneously, the effect of a second DMA-channel will be very good.

7 USE OF MEMORY IN AN ND-500 SYSTEM

As shown, ND-100 and ND-500 have separate swap systems, and they will each reserve a certain number of pages for swapping:



SEPARATE
SWAP SYSTEMS

This is a static allocation, but it can be changed (checked) by:

```
@ND-500-MONITOR
     N500: GIVE-N500-PAGES 0
     NO OF PAGES AVAILABLE FOR ND-500 PROCESSES    :  796
     NO OF PAGES USED BY SWAPPER PROCESS           :   44
     NO OF PAGES AVAILABLE FOR SWAPPING IN SIN-III :  456
```

The information from this command tells us how much memory each CPU has available for swapping. NUMBER OF PAGES USED BY

SWAPPER PROCESS is the code and data area size occupied by the 500 swapper (fixed in MPM memory, as the swapper is always the privileged ND-500 process number 0).

Tuning should be done so that the ND-100 part of the system has just sufficient memory not to swap significantly (swapping less than approximately 10% as measured by RT-PROGRAM-LOG). The rest of the memory available for swapping will then be used by the ND-500 CPU.

==========

7.1 MEMORY ALLOCATION

The default 100/500 memory allocation at start up, from the I-version and onwards of SINTRAN III, is as follows:

N = total no. of pages for swapping (local + MPM)

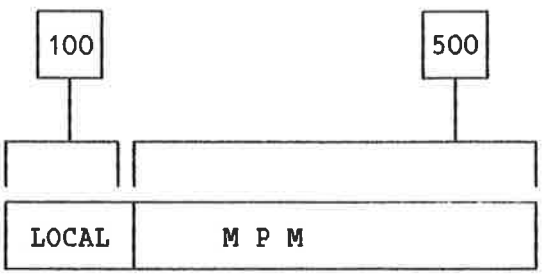
	100	500
N < 2 MB	N/2 MB	N/2 MB
2 MB < N < 4 MB	1 MB	N - 1 MB
4 MB < N < 8 MB	1.5 MB	N - 1.5 MB
N > 8 MB	2 MB	N - 2 MB

==========

7.2 ND-500 MEMORY CONFIGURATION

The physical memory configuration of any ND-500 system may be set up as one of these three possibilities:

1. Sufficient local memory



This is the case where ND-100 operates almost all the time in its own local memory. It is the ideal case, and the size of this local memory can be calculated from the formula in the next chapter.

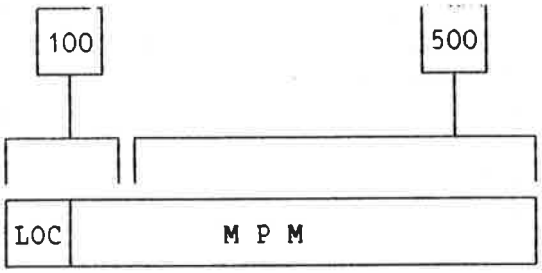
The advantage of having sufficient local memory in the system is twofold:

- Local memory is about twice as fast as MPM (seen from the ND-100)
- There will be no memory contention between ND-100 and ND-500

In effect, this means the ND-100 CPU runs as fast as possible when its memory references go to local memory.

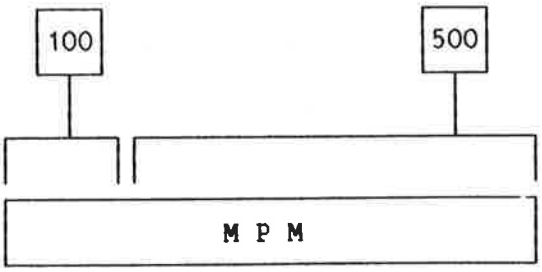
If we compare a situation with no local memory to one with sufficient local memory, the possible speed difference (throughput) is in the 5-30% range. A typical gain will be around 15%.

2. Some local memory



This is the most common situation for installed ND-500 systems. More local ND-100 memory should be added to the system.

3. No local memory



Some systems look like this. Local ND-100 memory should be installed.

8 MEMORY USED BY ND-100 IN AN ND-500 SYSTEM

The memory which is used by the ND-100 CPU in a 500 system should be installed as local ND-100 memory. This chapter describes how to arrive at an estimated size of this memory when running SINTRAN versions up to J. For K-version and onwards the sizes will increase.

8.1 MEMORY SHARED BY ALL USERS

This table shows which parts of the operating system and the system programs that should be allowed to stay in memory (the main work-set):

	Full size(MB)	Workset(MB)
SINTRAN FIXED	0.250	0.250
FILE SYSTEM (SHARED)	0.076	0.076
ND-500-MONITOR, CODE (SHARED)	0.084	0.084
ND-500-MONITOR, DATA (SHARED)	0.256	0.040
COMMAND SEGMENT	0.052	0.020
SPOOLING, ONE PROCESS RUNNING	0.032	0.032
REMOTE FILE ACCESS (OUT)	0.088	0.044
FILE SERVER (IN)	0.256	0.054
		= 0.6 MB

We have allowed for one spooling process running, and also for the use of remote file access and file server.

8.2 MEMORY PER USER

For each active 500 user, we must allow space for:

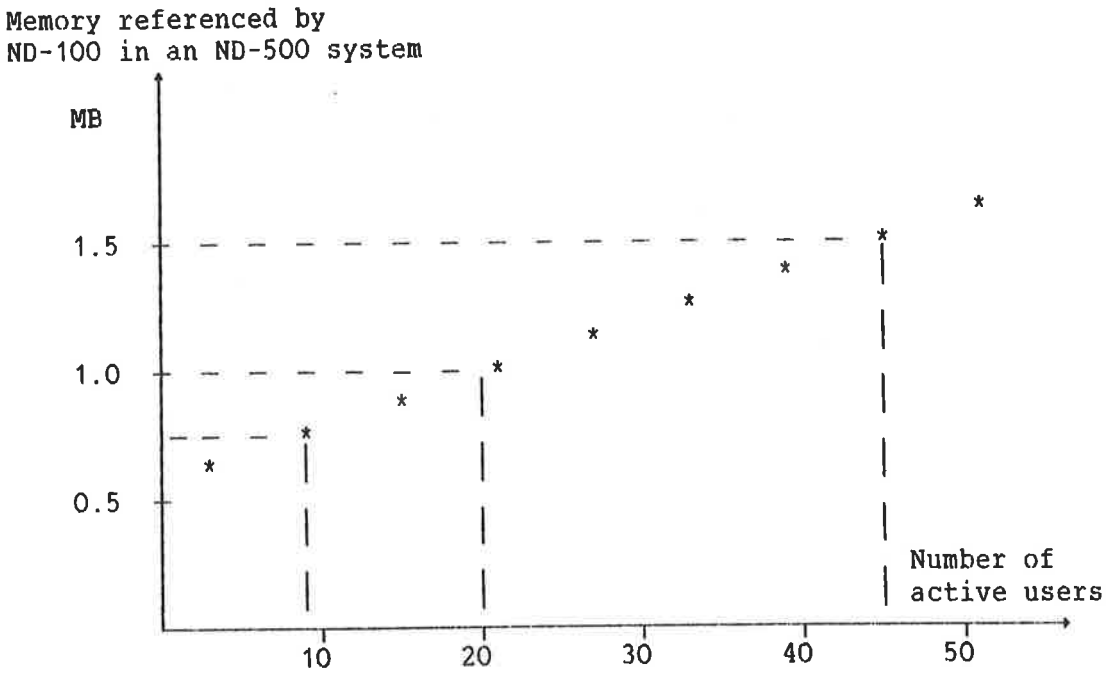
SYSTEM SEGMENT	: 0.010 MB
DATA-SEGMENT FOR ND-500	: 0.010 MB
	0.020 MB

TOTAL: **0.6 + N x 0.02 MB**

N = number of users

Observe that this formula allows for no ND-100 applications.

8.3 GRAPH

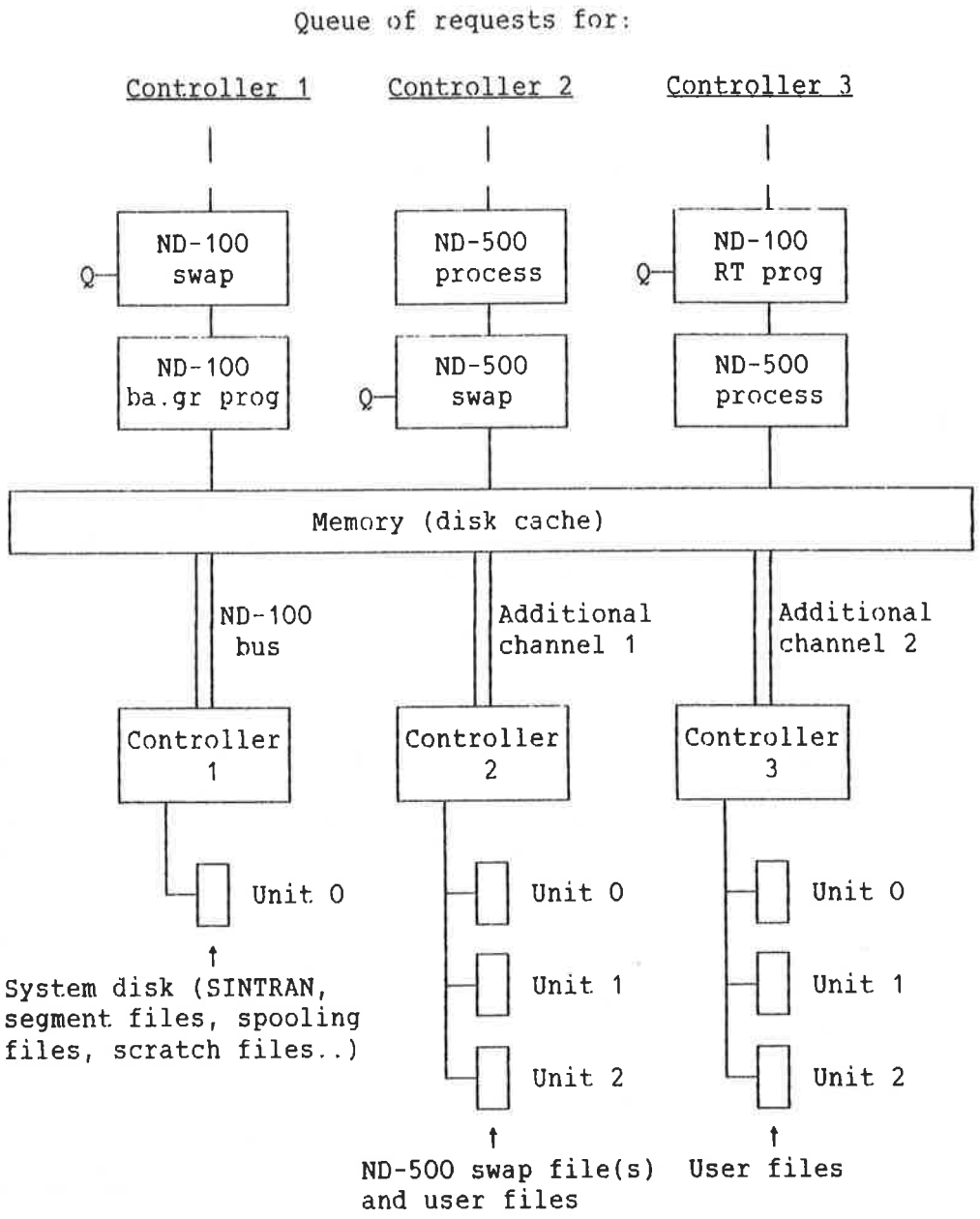


We can see that 1 MB of local memory should be used to support 20 active users.

9 EXAMPLE OF I/O-SYSTEM CONFIGURATION

The following example is meant to illustrate an I/O-system configuration for maximum throughput in an ND-500 system. Three disk controllers have been used to show all possible tasks that may be run in parallel. The controllers are run in parallel, except for the fact that the disk driver code is executed in the ND-100 CPU. This driver cost is, however, only around 5 ms CPU per DMA access. Also, queues have been indicated at single thread points in the system (by the letter Q).

Three different jobs are being served simultaneously in the example. One is an ND-100 background job using controller 1, the second is a 500 job causing swapping on controller 2, and the third a 500 process accessing controller 3 (could be direct transfer to/from contiguous files).



- ND-100 swapping and ND-500 swapping will run in parallel if the ND-100 segment file(s) and the ND-500 swap-file(s) are on different controllers, as shown. Note that they are both single thread, as indicated by the letter Q. This means that if more than one job of this type appears, it will have to queue up.

- I/O from RT-programs in ND-100 may run completely in parallel with swapping in 100/500, if the files accessed are on a third controller, as shown. More ND-100 RT-programs requesting I/O will have to queue up, due to the common stack for RT-programs in SINTRAN (I/O goes through RWRTx).

- ND-100 background programs doing I/O will run completely in parallel if on different controllers. So will ND-500 processes (time shared and RT processes).

●●

10 TIME SLICING IN SINTRAN III

10.1 GENERAL

A resident RT program changes the priorites of processes according to the CPU time they are using, in order to share the CPU resources between them. This program is called the time slicer program, RTSLI.

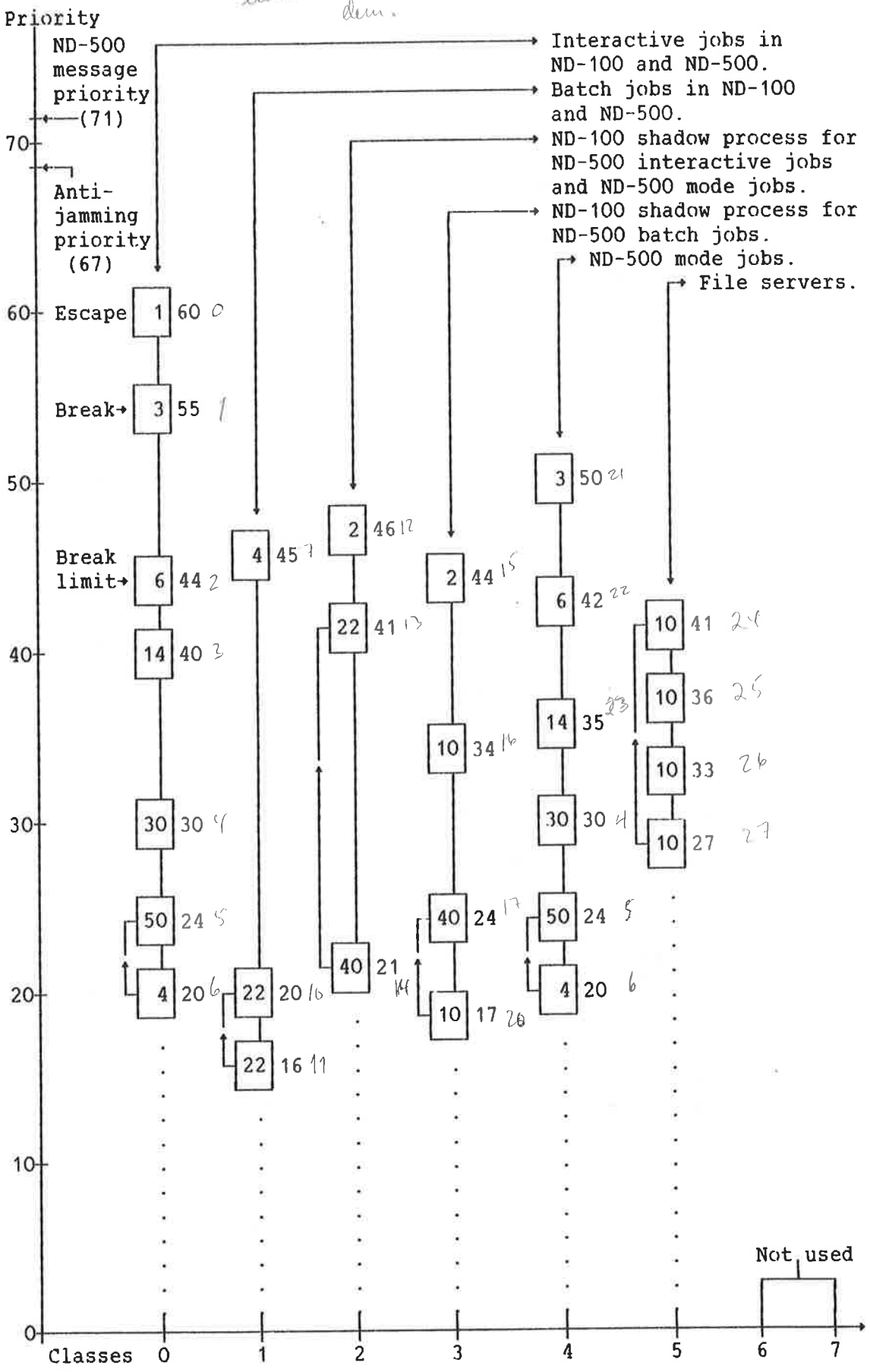
By default, all ND-100 background programs and all ND-500 processes - including batch - are time sliced. RT programs in ND-100 cannot be time sliced. RT processes in ND-500 will be time sliced, unless the SET-PRIORITY command (with a priority different from 0) has been executed for that process. The priority of a process, and the CPU time it can consume on that priority, are defined by the characteristics of the time slice class to which the process belongs.

There are 8 (0-7) time slice classes in the system, of which 6 (0-5) are used by the system. Classes 6 and 7 are free, and the use of these can be defined by SINTRAN-SERVICE-PROGRAM (DEFINE-TIME-SLICE).

A process running in ND-500 has two priorities, one in ND-500 and one in ND-100. The same algorithm is used for time slicing in ND-500 as in ND-100, except that ND-500 CPU time is used instead of ND-100 CPU time. The ND-100 priority of an ND-500 job is mainly used when the ND-100 "shadow" process executes monitor calls on behalf of the ND-500 process.

10.2 ILLUSTRATION

The diagram on the next page illustrates the time slicing mechanisms. Note that all numbers are in octal format!


Priority
ND-500 message priority (71)
Anti-jamming priority (67)
Interactive jobs in ND-100 and ND-500.
Batch jobs in ND-100 and ND-500.
ND-100 shadow process for ND-500 interactive jobs and ND-500 mode jobs.
ND-100 shadow process for ND-500 batch jobs.
ND-500 mode jobs.
File servers.
Escape
Break
Break limit
Not used
Classes
70
60
50
40
30
20
10
0
0
1
2
3
4
5
6
7

ND-30.049.1 EN

==========

10.3 ADDITIONAL INFORMATION

The previous diagram illustrates the time slice mechanism included as standard in the I-version of SINTRAN III Release Information. Note that:

- The number of time slices on each priority level is given inside the square in the diagram, the priority itself to the right of it.

- The time slice process (RTSLI) runs twice a second.

- One time slice is by default equal to 240 ms (decimal) of CPU (ND-100 or ND-500 CPU time).

- A process is reset to break priority (55B) only if current priority is less than 44B (break limit) when a break condition is met.

- In interactive mode, a process is reset to escape priority (60B) if an escape is typed, when escape is enabled.

- Due to the fact that the time slicer program (RTSLI) is running at an interval of 0.5 seconds, a delay of max 0.5 seconds can occur when typing a break or escape character, before the priority is raised.

- The number of time slices on each priority level has been slightly adjusted from the I-version to the J-version. (J-version values are shown.)

- Classes for ND-500 mode jobs and for file server (TADXX) were included from the J-version.

- Manipulation of time slice parameters is done via SINTRAN-SERVICE-PROGRAM (standard from J-version, patch to I-version).

- Processes always move steadily to a lower priority, except from break conditions, back arrows (from the lowest priority), antijamming priority (67B) conditions for system semaphores in ND-100, and message priority conditions (71B, between 100/500) for ND-500 jobs. A system semaphore is defined as a datafield with the protection ring value (in the Typring location) set to 2 or 3. Examples of messages from the ND-500 to the ND-100 are examine/deposit ND-500 registers, read/write monitor call parameters. The ND-500 message priority is used only in the ND-500.

- Antijamming priority and message priority are only used for very short periods, until the semaphore is released or the message handled. The priority is then reset to its original value. These special priority changes are made "outside" the time slicer, and they are done whenever needed, and not at the fixed 0.5 second interval. Antijamming is only used if a queue (two requests or more) starts to build up at a certain resource, and not when just one job requests a resource.

- In ND-500, the priority of a process is increased by one for each monitor call executed by the process, provided that current priority is in the range lowest to second lowest for the class of the process. This is done to distribute resources evenly between heavy jobs.

- A special algorithm has been included to avoid a situation where the time slicer becomes too "stable" (phasing). This algorithm checks for the number of time slices to be consumed at the current priority level. If this number is greater or equal to 22B, then a random number of time slices in the interval 0B - 17B is added to the number of slices which can be consumed.

11 SYSTEM PARAMETERS FOR ND-500

Use the ND-500-MONITOR command LIST-SYSTEM-PARAMETERS to check the current values of these parameters. Starting with version J of SINTRAN III, the following system parameters are valid:

NO-OF-PHYSICAL-SEGMENTS
CLEAN-SEGMENT-N-PF
SWAPOUT-SEGMENT-N-PF
DISC-CACHE-BUFFER-SIZE
NUMBER-OF-DISC-CACHE-BUFFERS
MAX-PAGES-FIXED
DEFAULT-EXTRA-PAGES-TO-ND-500

NO-OF-PHYSICAL-SEGMENTS is just the maximum number of physical segments that can be used at any given time. It must be increased if the error message NO FREE PHYSICAL SEGMENT appears.

The parameters CLEAN-SEGMENT-N-PF, SWAPOUT-SEGMENT-N-PF, DISC-CACHE-BUFFER-SIZE and NUMBER-OF-DISC-CACHE-BUFFERS have to do with the internal workings of the ND-500 swapper. See the manual ND-500 Loader/Monitor for more information.

MAX-PAGES-FIXED is the maximum number of pages that can be fixed in the 500 memory at any given time. The default value is approximately half the size of the available memory for ND-500 swapping. If some user tries to exceed this limit, the system will have to start unfixing other pages, if possible.

DEFAULT-EXTRA-PAGES-TO-ND-500 adjusts the allocation of memory between 100/500 if used, as shown in the chapter "Use of memory in an ND-500 system". Normally, it is easier to insert a GIVE-N500-PAGES or TAKE-N500-PAGES in the LOAD-MODE file to achieve the same result.

12 PRIORITIES FOR SYSTEM PROGRAMS

The following is a list of normal priority levels for different tasks in a system. An arrow indicates that the default priority has been adjusted from the C or E release of the product (see below).

	PRIORITIES (DECIMAL)
SPRTX	44
COSPO	44
BAKXX, TADXX	16-20-24-32-36-45-48
BCHXX	14-16-37
XFTRA	56 ⟶ 22 (C-release)
SIB2A,.. for ND-500	46 ⟶ 33 (E-release)
SIB2A,.. for ND-100	39 ⟶ 33 (E-release)
SERV2A,..	40 ⟶ 32 (E-release)
FILE-SERVER (TADXX)	23-27-30-33

These priorities should fit most standard systems. But they may of course be changed to meet special requirements.

13 (SYSTEM) PROGRAMMING

13.1 ND-100

ND-100 background users do not have the same direct transfer possibilities as for ND-500. Only RT-programs can fix the necessary segments in memory, and do I/O to/from these segments directly (SIBAS is a good example). I/O from background programs will always have to go through the file system, and therefore use the internal buffer (cache) of the file system. This buffer can be up to 64 pages big, and the size is decided when SINTRAN is generated. The size can be checked by the (octal) value of the symbol 5BUFA found in the file SYMBOL-2-LIST under user SYSTEM.

If the user program requests more than one page (2 KB) in one I/O-operation, the file system will chop it up into physical accesses of one page at the time.

13.2 ND-500

13.2.1 FORTRAN I/O FROM ND-500 PROGRAMS

There are a number of ways to do file I/O from ND-500 FORTRAN. The "file as segment" method has been mentioned before. It is available from ND-500 FORTRAN via an OPEN statement of the following type:

```
OPEN(.....,MODE='SEGMENT',.....)
```

When a file is opened for ordinary I/O, the access parameters "R" and "W" should never be used. The compiler will warn against these.

In case of sequential I/O to contiguous files, there are two methods that can be used to speed up I/O considerably. Both are based on facilities in the ND-500-MONITOR for fixing contiguous areas in physical memory, in order to use DMA on large blocks. The first method is available when the monitor calls RFILE and WFILE are used directly in the program. If

the file has been opened with

```
OPEN(.....,ACCESS='SPECIAL',.....)
```

the pages affected by RFILE or WFILE calls will be fixed (contiguously) by the ND-500-MONITOR. The fixing takes place when the first RFILE or WFILE call is encountered, and may involve a full data segment or only a part of it. Every time an RFILE or a WFILE to a logical address area occurs, the monitor fixes that area if it has not done so before. In this way, a large part of available physical memory may be fixed on behalf of a single 500 process. Since only a certain portion of the 500 memory can be fixed, there may not be memory available for fixing, and then the I/O calls will be broken down to the customary 2 KB (1 page).

The fixing and unfixing of memory pages are time consuming task. This should be taken into consideration when using this method. In other words, one should try to do many DMA operations to and from the same fixed data area to get a good effect. Unfixing takes place upon return from an RFILE or a WFILE call if the monitor finds out that too much memory has been fixed, i.e. the system parameter MAX-PAGES-FIXED has been exceeded. If this limit is not exceeded during execution, unfixing takes place upon exit from the ND-500-MONITOR.

Example:

```
      PROGRAM FASTIO
      ...
      INTEGER TABLE(8192)
C  This array has a size of 32 KB, since an integer on ND-500 is
C  32 bits.
...
...
OPEN(INFILE,...,ACCESS='SPECIAL'.....)
...
...
CALL RFILE(INFILE,0,TABLE(1),IBNO,NWORDS)
C  The second parameter is set to 0, which means that the process
C  will be suspended until the transfer is complete. The "nowait"
C  mode may also be used. The third parameter
C  points to the memory address of the first data element to be
C  transferred, the fourth to block number on the file, and the last
C  parameter specifies the number of words to be transferred.
...
...
```

Another method is available from FORTRAN version I. This

method is used for ordinary FORTRAN READ and WRITE, and uses a fixed I/O buffer inside the FORTRAN I/O-system. I/O to a certain data area in the program will then involve copying to or from the I/O buffer. The amount of data transferred in an actual disk access depends on the specification of BUFFER_SIZE in the OPEN statement for the file involved (number of bytes). If no BUFFER_SIZE is specified, the SINTRAN III page size (2 KB) is used.

Example:

```
PROGRAM BUFFIO
...
INTEGER TABLE1(2048)
INTEGER TABLE2(4096)
INTEGER TABLE3(1024)
...
...
OPEN(IFILE1,...,ACCESS='SPECIAL',BUFFER_SIZE=8192,.....)
OPEN(IFILE2,...,ACCESS='SPECIAL',BUFFER_SIZE=16384,....)
OPEN(IFILE3,...,ACCESS='SPECIAL',BUFFER_SIZE=4096,.....)
...
...
READ(IFILE)TABLE1
...
READ(IFILE)TABLE2
...
READ(IFILE)TABLE3
...
...
```

When the program is loaded, one must use the LINKAGE-LOADER command:

```
L-L: SET-IO-BUFFERS <NUMBER-OF-BUFFERS>
```

The number of I/O buffers must correspond to the total buffer size needed for DMA. For instance, if one needs to operate simultaneously on the three files in the example, the number of IO-buffers should be at least (4+8+2) 14 pages.

If the command:

```
L-L: LINK FORT-LIB
```

is used, or none at all (automatic linking), then a certain number of I/O-buffers will automatically be allocated. This number is system dependent, but is usually set to 30.

The main difference between the "RFILE/WFILE" method and the "BUFFER_SIZE" method is that the former uses dynamic memory fixing directly on the data area(s) involved, while the latter is based on a fixed I/O buffer of preset size, and

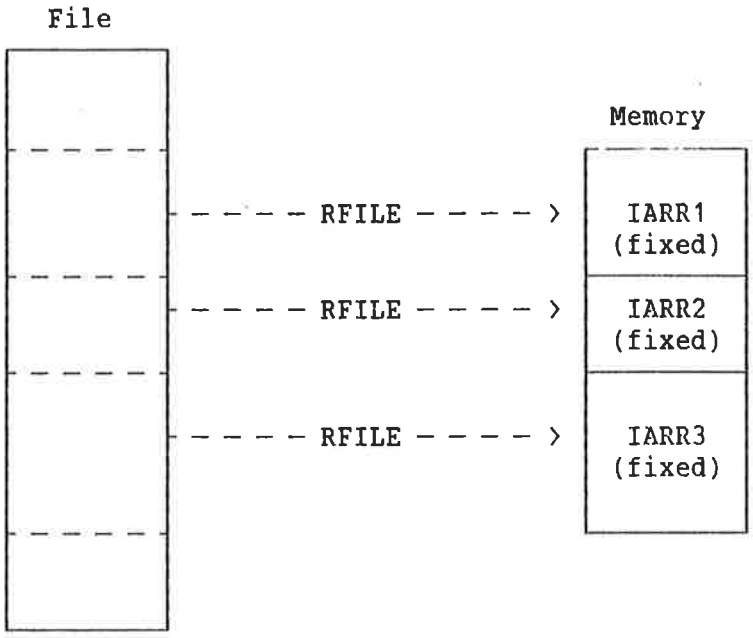
copying between the buffer and the data areas in connection with each I/O operation. The cost of such copying (approximately 2 ms ND-500 CPU-time per KB in an ND-570) must be weighed against the possible gains from doing I/O on large blocks.

However, one can of course use a similar mechanism for the "RFILE/WFILE" method, in that all physical I/O can be routed through an auxiliary data area which serves as a buffer. The various alternatives are illustrated with examples in the following. It is of course assumed throughout that the involved files are contiguous.

Example:

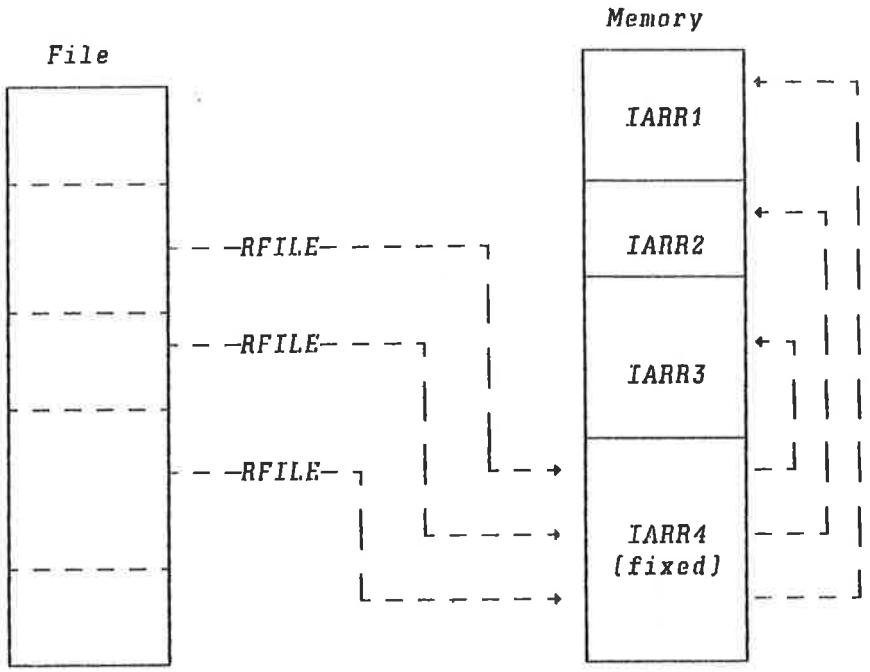
Method 1:

Use of RFILE or WFILE directly on relevant data areas. This implies that the monitor will try to fix a lot of memory. This may be desirable if many I/O operations from the same memory areas are involved, and will then secure extremely fast I/O. But there may be a good deal of overhead in connection with fixing and unfixing.



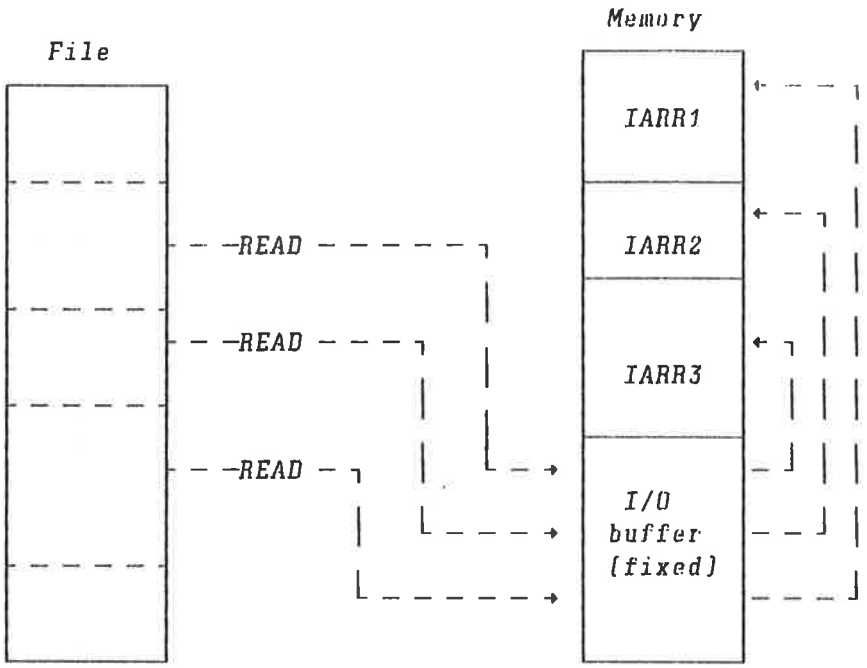
Method 2:

Use of RFILE or WFILE on an auxiliary data area only, and copy to or from appropriate data areas.



Method 3:

Use of buffered I/O with FORTRAN READ and WRITE.



Methods 2 and 3 will fix less memory than method 1. However these methods will need more address space because auxiliary data buffers are introduced.

14 TUNING OF SIBAS SYSTEMS

14.1 BOTTLENECKS IN SIBAS SYSTEMS

The first step in a tuning study of a SIBAS system should be to get information on the loads on the various resources in the system, and determine which resource is the bottleneck. The most important resources are CPUs, disks and SIBAS itself (since SIBAS can only serve one request at a time).

The most common bottleneck in an ND-100 system running SIBAS, is the CPU. In singlemachine SIBAS 100 systems with much screen I/O, it is practically certain that the CPU is the

bottleneck. This is because screen I/O in a transaction processing system usually is considerably more CPU-consuming than SIBAS itself.

At the other extreme, we have SIBAS ND-500 back-end systems, where SIBAS, as a shared logical resource, frequently will be the bottleneck. If SIBAS processing is the only significant activity in the 500 CPU, the CPU will mostly stay idle while SIBAS is in I/O wait for a disk transfer.

Example:;
This example illustrates how you can determine the bottleneck in a SIBAS 500 transaction processing system. Suppose that typical loglines from PROCESS-LOG-ALL read as follows:

```
      PROCESS LOG ALL    12.39. 3      29 NOVEMBER 1984

0   1   2   3   4   5   6   7   8   9  10  11  OTHERS  IDLE  SWAP

6  30   0   1   0   0   0   1   1   0   0   0     3      58    60
6  28   0   0   0   1   0   1   0   0   0   0     4      60    62
```

Corresponding output from RT-PROG-LOG could be (no RT program specified):

```
CPU   SWAP  FILES DISK  UNIT 1207
 45    00    49    00     49
 43    00    47    00     47
```

The database disk (logical unit 1207) appears to be the most heavily used resource in the system, with about 48%. The utilization of the ND-100 CPU is about 44%, while the utilization of the ND-500 CPU is about 41%. Thus the system seems well balanced, and you might think that it could be even heavier loaded. However, a system like this may well produce long response times! We have to look at the load on SIBAS. Assuming that there is no genuine swapping in the ND-500, the load on the ND-500 swapping system is entirely due to "file-as-segment" swapping generated by SIBAS. Therefore, the total load on SIBAS will be:

```
   CPU load from ND-500 swapper:      6 %
+  CPU load from SIBAS 500:          29 %
+  SIBAS I/O wait (SWAP in the log): 61 %
   Total load on SIBAS:              96 %
=========================================
```

Thus the diagnosis is entirely clear. SIBAS itself is the bottleneck.

In the next sections, we discuss various methods that can be used to enhance the performance of SIBAS systems.

14.2 SEVERAL SIBAS SYSTEMS

If the data structure is such that it can be split into two or more databases (with one SIBAS system for each), one can reduce or eliminate the limiting effect of SIBAS as a logical resource (illustrated in the example above). In short, one SIBAS system can use the CPU while another waits for a disk transport. To get any significant effect, there must be a suitable distribution of loads on the different SIBAS systems.

Actually, many of our SIBAS customers already run their systems in this manner, not because of performance considerations, but because splitting the total data structures in this way is natural.

14.3 SIBAS MACROS

User written SIBAS macros by means of the SEXMC call is a little used but highly advantageous method of reducing CPU loads in SIBAS systems. The principle is that a sequence of logically related SIBAS calls are collected in a single SEXMC call. SIBAS will be reserved for the entire SEXMC call.

A very important benefit from this is that the "back and forth" traffic between SIBAS and the various applications will be reduced, and may thus reduce the CPU overhead substantially. In SIBAS backend systems this may be of enormous importance, because frequent SIBAS calls may overload the ND-100 CPU with communication.

Generally, the communication cost of a single SIBAS call is between 15 and 20 ms in the ND-100 CPU in the case of a local area network, and between 100 and 150 ms in a public network like X.21. These costs depend very little on the message lengths, so by using a SIBAS macro containing 5 SIBAS calls, the communication cost will be reduced to about one fifth. In comparison, the CPU cost for the processing of a single SIBAS call varies from an average of about 20 ms on an ND-100/CX to about 7 ms on an ND-570/CX.

Another important effect is the use of SIBAS work area. Each SIBAS call needs a certain data context, which SIBAS must set up when processing the call. If the application contains a sequence of logically connected calls that work on the

same data, it will of course be an advantage if SIBAS can retain the data for the whole sequence without being disturbed. For ND-100 SIBAS, this will even save disk accesses, because data that has been thrown out of SIBAS work area at some time, must be retrieved from the disk if needed again. However, SIBAS macros should not be made too long, in order that short transactions are not unduly delayed.

==========

14.4 DATABASE STRUCTURE

The time needed for a SIBAS call depends heavily on the database structure. In general, many indexes and sets may cause a very high number of disk accesses. A STORE call to a realm with N indexes and M doubly linked sets may generally cause

$$2 + 4 \times N + 4 \times M$$

disk accesses: There will be one access to read in the page on which the record will be stored, and one to write it back. 3 index levels may be read and the lowest written back. The successor and predecessor in each set must be read to have their set pointers updated and written back. So there is a clear trade-off between functionality and number of access ways on one hand and performance on the other. If most of the on-line activity is read only, then such complexity may not be of great importance, but if there is heavy updating, it may have a disastrous effect on performance.

==========

14.5 MEMORY CONSIDERATIONS

The "file-as-segment" concept introduces the possibility of adding physical memory to reduce disk I/O in ND-500 SIBAS systems (there is no such effect on ND-100 SIBAS). In SIBAS D, file-as-segment is automatically used for all database files that do not exceed the maximum segment size of 132 MB. Big files like this will be opened for ordinary file I/O. From the E version of SIBAS it is optional whether a database file should be opened as a segment or as an ordinary file.

Generally, data frequently accessed should stay in memory as much as the swapping system will allow, whereas big files with low access rates for each individiual record should not be opened as segments. The reason for this is that the

file-as-segment option will give that particular segment a certain amount of space in memory, while it is unlikely that the data on it will be accessed again within a short period of time. Therefore, one might as well free this memory space by using ordinary file access.

==========

14.6 DISTRIBUTION OF DATABASE ON SINTRAN FILES

In SIBAS 100 systems it may be an advantage to keep a data realm and its index tables on the same SINTRAN file, since this may reduce both seek times and the number of disk accesses (reduced number of index block accesses). For SIBAS 500, other factors must be considered. Due to "file-as-segment", one should try to assess which data will be most frequently accessed, relatively to the memory size. The distribution of data and index tables on SINTRAN files should reflect the frequency of use, so that often used and rarely used data should preferably be stored on different SINTRAN files.

==========

14.7 SIBAS LOGGING

SIBAS uses two different types of logging: Routine log (often called "call-log" or "transaction log") and BIM log (Before-Image-Log). The routine log registers all updates that have taken place after a certain "initial" state, which may be defined as the state of the database immediately after installation of a correct backup or after a SIBAS checkpoint.

The BIM-log collects the originals of the database pages that are updated after a checkpoint. If a "crash" occurs, the BIM-log can be used for a "Roll-back" to the "initial" state (the state at the checkpoint), and then the routine-log is used to "re-process" the database. To minimize logging, answer "N" to the five first questions in NEW RUNFLAG? and "Y" to the last.

15 MISCELLANEOUS

15.1 SCRATCH FILES

The default size of the scratch file (when the user logs out) has been increased from 32 pages in the I-version of SINTRAN to 64 pages in the J-version. This is very useful if relatively large files are being worked on, as no (or few) new pages has to be allocated for the scratch file during the terminal session.

15.2 USE OF FILES IN GENERAL

Contiguous files should be used whenever possible, as no index pages have to be read/written for these files. Go through the most commonly used files in the system (check the number of times they have been opened), and see if they can be copied to contiguous files.

Allocating new pages to an expanding indexed filed is a very heavy file system operation; it takes around 4 - 5 times as much resources as using existing pages on a file.

15.3 BUFFERED TERMINAL INTERFACE

Let us look at the performance implications of buffered terminal interface boards (ND 273 and ND 274):

- It requires the J-version or later of SINTRAN III.
- Buffered and nonbuffered interfaces can be mixed in a system. SINTRAN III will find out which is which, and use the optimum driver strategies.
- The interface buffers up to 16 characters on input and 64 on output per line.
- The ND 274 variant can run at line speeds of up to 19200 baud, ND 273 up to 9600 baud.

The performance improvements are realized by the fact that the ND-100 CPU-cost for driving characters to a terminal (output driver) is reduced. The input driver cost stays the same as before.

==========

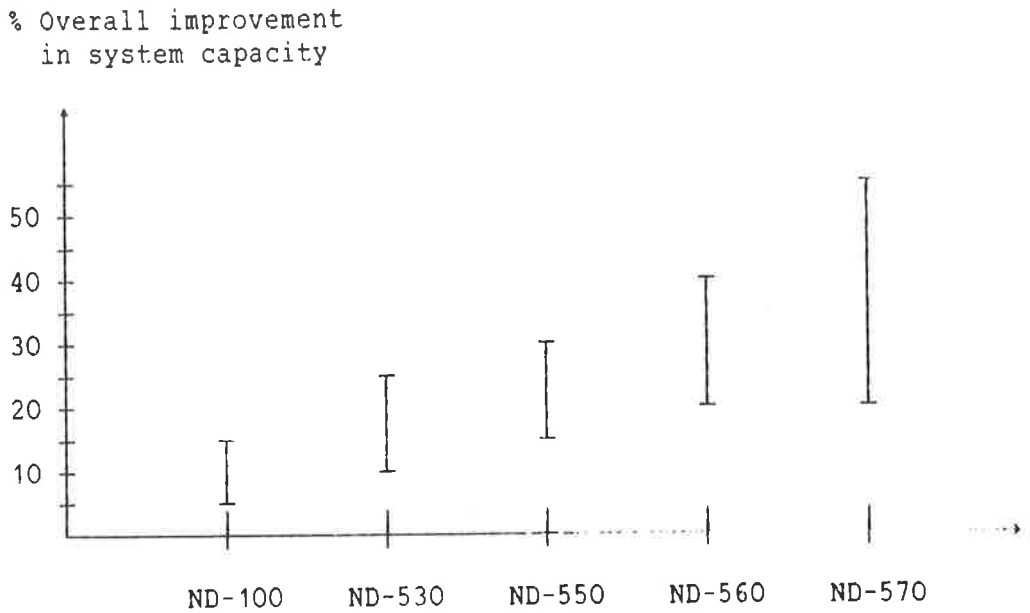
15.3.1 ND-100

If we consider an ND-100/CX system (or standard ND-100) used mainly for screen-oriented tasks (NOTIS products, PED, FOCUS, NSHS, ...), the overall CPU improvement by using buffered interface cards will always be in the 5-15% range, and generally around 10%. For less "interactive" systems, the overall CPU improvement will be in the 0-5% area.

==========

15.3.2 ND-500

For ND-500 the picture is more complex, but the improvement potential is much larger. The greatest improvement will be seen in a "screen-oriented" 500-system where the ND-100 CPU is the bottleneck. This is the case for many ND-500 systems.



This diagram illustrates approximately the overall capacity improvements that can be expected in systems of the screen-oriented type (frequent screen output).

For ND-500 systems that are CPU-bound (ND-100 CPU) or disk bound (use of disk is higher than use of ND-100 CPU), the effects will be marginal.

********** SEND US YOUR COMMENTS!!! **********

Are you frustrated because of unclear information in this manual? Do you have trouble finding things? Why don't you join the Reader's Club and send us a note? You will receive a membership card — and an answer to your comments.

Please let us know if you

- find errors
- cannot understand information
- cannot find information
- find needless information

Do you think we could improve the manual by rearranging the contents? You could also tell us if you like the manual!

********** HELP YOURSELF BY HELPING US!! **********

Manual name: SINTRAN III Tuning Guide

Manual number: ND–30.049.1 EN

What problems do you have? (use extra pages if needed) _____

_____

_____

_____

_____

Do you have suggestions for improving this manual ? _____

_____

_____

_____

_____

_____

Your name: _____ Date: _____

Company: _____ Position: _____

Address: _____

_____

What are you using this manual for ? _____

_____

NOTE!
This form is primarily for documentation errors. Software and system errors should be reported on Customer System Reports.

Send to:
Norsk Data A.S
Documentation Department
P.O. Box 25, Bogerud
0621 Oslo 6, Norway

Norsk Data's answer will be found on reverse side

Answer from Norsk Data _____

_____

_____

_____

_____

_____

_____

_____

_____

_____

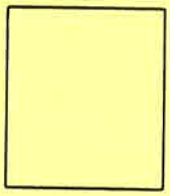
_____

_____

_____

_____

Answered by_____ Date _____



Norsk Data A.S
Documentation Department
P.O. Box 25, Bogerud
0621 Oslo6, Norway

Systems that put people first

NORSK DATA A.S OLAF HELSETS VEI 5 P.O. BOX 25 BOGERUD 0621 OSLO 6 NORWAY
TEL.: 02 - 29 54 00 - TELEX: 18284 NDN