

Reliant UNIX 5.45

Tuning Guide

Comments... Suggestions... Corrections...

The User Documentation Department would like to know your opinion on this manual. Your feedback helps us to optimize our documentation to suit your individual needs.

Fax forms for sending us your comments are included at the back of the manual.

There you will also find the addresses of the relevant User Documentation Department

Copyright and Trademarks

Copyright © 1999 Siemens AG.

All rights reserved.

Delivery subject to availability; right of technical modifications reserved.

All hardware and software names used are trademarks of their respective manufacturers.

This manual is printed on
paper treated with
chlorine-free bleach.

Introduction

Possible system bottlenecks

Tools for measuring performance

Shell procedures for load measurement

Evaluating the load measurements

Implementing tuning measures

Configuring the kernel

Appendix

Glossary

Abbreviations

Figures

Tables

Related publication

Index

Contents

1	Introduction	1
1.1	Target group	1
1.2	Design of the manual	2
1.3	Notational conventions	3
2	Possible system bottlenecks	5
2.1	Characteristics of the storage media	6
2.2	Processors	7
2.3	Main memory	7
2.3.1	Main memory segments	10
2.3.2	seg_kmem segment	11
2.3.3	segu segment	13
2.3.4	segmap segment	14
2.3.5	Transferring main memory pages	15
2.3.6	Page cache	15
2.3.7	Shared memory alignment	16
2.3.8	Big pages	16
2.3.9	RM600 E: Software support for ccNUMA	21
2.4	Hard disks and file systems	24
2.5	System tables and kernel parameters	26
3	Tools for measuring performance	29
3.1	Summary	29
3.2	sar(1)	31
3.3	sysadm(1M)	34
3.4	dkstat(1M)	37
3.5	mpstat(1M)	39
3.6	vmstat(1M)	41
3.7	scstat(8)	45
3.8	iotail(1M)	47
3.9	tunestat(1M)	48
3.10	truss(1)	49
3.11	proceso(1M)	51
3.12	xload	55
3.13	SINIX/monitor	57
3.13.1	Main window	60
3.13.2	Detail windows	62
3.13.3	Options window	64
3.13.4	Help window	67

Contents

- 4 Shell procedures for load measurement 69**
 - 4.1 Overview of shell procedures 69
 - 4.2 Shell procedure ps_meas 70
 - 4.3 Shell procedure vm_meas 72
 - 4.4 Shell procedure dk_meas 72
 - 4.5 Shell procedure main_meas 73

- 5 Evaluating load measurements 77**
 - 5.1 Hardware configuration 77
 - 5.2 Operating system version 80
 - 5.3 Information about processors 80
 - 5.3.1 Evaluating CPU load 83
 - 5.3.2 Evaluating process queue statistics 84
 - 5.3.3 Determining system status of individual processes 85
 - 5.3.4 Identifying CPU-intensive processes 87
 - 5.4 Analyzing the main memory resource 89
 - 5.4.1 Characteristic main memory sizes 90
 - 5.4.2 Size of free memory 91
 - 5.4.3 System table utilization 93
 - 5.4.4 Buffer cache load 95
 - 5.4.5 Kernel memory allocation 96
 - 5.4.6 Paging activity 98
 - 5.4.7 Swapping activity 102
 - 5.4.8 Process memory requirements 104
 - 5.4.9 Analyzing the swap area 107
 - 5.5 Hard disks and file systems 109
 - 5.5.1 Identifying installed hard disks and SCSI controllers 109
 - 5.5.2 Displaying/modifying the partition layout of a hard disk 110
 - 5.5.3 Identifying partitions used as file systems 111
 - 5.5.4 Determining the fill level of file systems 112
 - 5.5.5 Checking whether the CPU is waiting for I/O 113
 - 5.5.6 Determining the disk load 114
 - 5.5.7 Caching file I/O 118
 - 5.5.8 Determining file access frequencies 120
 - 5.5.9 Analyzing inode utilization in file systems 122
 - 5.5.10 Analyzing system calls 123
 - 5.6 System tables 125
 - 5.6.1 Analyzing the process table, i-list (inode table) and file table 125
 - 5.6.2 Analyzing message queues 126
 - 5.6.3 Analyzing semaphores 128
 - 5.6.4 Analyzing shared memory 130

6	Implementing tuning measures	133
6.1	Processors	133
6.1.1	Binding processes to a CPU	133
6.1.2	Process scheduling	134
6.2	Main memory	136
6.2.1	Storing data for longer in main memory	136
6.2.2	Extending main memory	136
6.2.3	Reducing reserved shared memory	137
6.2.4	Increasing the swap area	137
6.3	Hard disks and file systems	138
6.3.1	Selecting virtual disk management	138
6.3.2	UFS and VxFS file system types	142
6.3.3	Minimizing program load operations	142
6.3.4	Identifying users who occupy considerable disk space	143
6.3.5	Searching for large files	144
6.3.6	Searching for old files	144
6.3.7	Relocating user data to free file systems	145
6.3.8	Reducing the time taken to search directories	146
6.3.9	Length of file and directory names	147
6.3.10	Modifying optimization strategies for UFS file systems	147
6.3.11	Determining the block size of a file system	148
6.3.12	Installing additional hard disks	148
6.3.13	Balancing hard disk utilization	149
6.4	System tables	149
6.5	Minimizing the number of processes	150
6.5.1	Deactivating unnecessary timing jobs	150
6.5.2	Deactivating uucp	150
6.5.3	Deactivating unrequired daemon processes	151
6.5.4	Deinstalling software which is no longer required	151
6.6	Optimizing compilers	151
6.7	TCP/IP and NFS tuning	152
6.7.1	The enhanced protocol stack	152
6.7.2	Performance of the enhanced stack	154
6.7.3	Effect of the socket options SO_SNDBUF and SO_RCVBUF and the routing metrics	154
6.7.4	Commands for resolving performance bottlenecks	154
6.7.4.1	netstat/enetstat	154
6.7.4.2	etherstat	159
6.7.5	Slot for zx subcontrollers	161
6.7.6	NFS and RPC	161

Contents

7 Configuring the kernel 163

7.1 The mtune file 163

7.2 Reasons for modifying the kernel parameters 165

7.2.1 Special requirements 166

7.2.2 Overwriting the operating system limit values 167

7.3 Determining the values of kernel parameters 168

7.4 Modifying the kernel parameters 172

7.4.1 Modifying the kernel parameters using idtune(1M) 174

7.4.2 Modifying the kernel parameters using SYSADM 174

7.5 Generating and booting a new kernel 176

7.5.1 Generating the kernel using idbuild(1M) 177

7.5.2 Generating the kernel using SYSADM 177

7.5.3 Booting the new kernel 178

7.6 Booting the old kernel 179

7.7 The rlimit structure 181

7.8 Description of the kernel parameters 181

7.8.1 General kernel parameters 183

7.8.2 File system parameters 188

7.8.3 Paging parameters 189

7.8.4 STREAMS parameters 189

7.8.5 Message parameters 191

7.8.6 Semaphore parameters 192

7.8.7 Shared memory parameters 193

7.8.8 Device driver parameters 194

7.8.9 Timer and scheduler parameters 195

7.8.10 Resource limit parameters 196

7.8.11 Specific parameters in Reliant UNIX 198

7.8.12 Computer-specific Parameters 202

8 Appendix 205

8.1 A brief introduction to tuning 205

8.2 Maximum values for sar analysis 206

8.3 Technical data for hard disks 210

Glossary 217

Abbreviations 237

Figures 241

Tables 243

Related publications 245

Index 251

1 Introduction

That the Reliant® UNIX® operating system offers good performance has been proved by a number of tests, so-called benchmark tests. You may, however, find that the performance of your system does not meet your expectations. There may be a number of reasons for this: the cause may lie in the applications which you are running or the hardware configuration of your computer may be inadequate.

This manual has three objectives:

1. It explains how you can analyze the performance in order to identify any bottlenecks.
2. It describes the measures you can take to improve performance (tuning).
3. It tells you when it is advisable to extend your hardware.

For a general description of how a UNIX system functions and the factors which influence performance, refer to the relevant specialist literature. In particular, the book “UNIX System V Release 4 - The Magic Garden Explained” by B. Goodheart and J. Cox is highly recommended. This manual in hand describes only those factors which are of significance for the operating system Reliant UNIX and it contains no descriptions of measures used to tune databases and networks. Both subject areas are so complex that they far exceed the scope of this manual. It was also not possible to examine the question of accounting. However, this subject is planned for future versions. For information on the VERITAS™ file system refer to the chapter “Performance and tuning” in the manual “VERITAS File System (VxFS)”.

The expectations placed on a tuning manual vary greatly. Please help us to improve this manual by telling us your comments, criticisms and experiences so that we can take these into account in subsequent editions. Use the page “Comments, Suggestions, Corrections” on the end of the manual.

1.1 Target group

This Guide is intended for experienced system administrators with a good knowledge of UNIX.

1.2 Design of the manual

Possible system bottlenecks are described together with examples showing how these bottlenecks can be identified.

The suggestions for improving system performance include simple measures such as modifying the disk partitions as well as system-specific measures such as the modification of various kernel parameters.

However, even the most intensive tuning of a system is no substitute for careful applications programming and well-conceived database design. These issues do not form part of these Guide.



Incorrect or incomplete tuning may result in the deterioration of system performance!

The objective of improving system performance can only be achieved if you consider the system as a whole. It is not enough simply to search for one bottleneck. You must identify all the bottlenecks.

You should always document changes to the system. Careful measurement under identical operating conditions (same applications before and after tuning) is also essential.

The chapter “Configuring the kernel” on page 163 describes the structure of the file `/etc/conf/cf.d/mtune` together with the most important tunable parameters. In most cases, the preset default values of the kernel parameters are satisfactory and it is therefore not essential to read this chapter. However, if you wish to modify one or more of these parameters, please make sure that you observe the comments in the chapter “Modifying the kernel parameters” on page 172.

1.3 Notational conventions

This manual uses the following notational conventions:

Italic type

Names of files, programs, commands, variables and options as well as screen text such as input fields, text fields and menus in the running text

Command(number)

The number following a command refers to the chapter in which the command is described in one of the following manuals:

- “Commands. User’s Reference Manual”
- “System Administrator’s Reference Manual”
- “Networking - Reference Manual”

The brackets and the number are not part of the command, i.e. they must not be entered.

Monospaced type

System outputs (error messages and other messages) and file entries

Bold monospaced type

User input in examples, with the exception of variables (see above)

“Quotes”

References to other chapters or manuals

, , 

Keys on the keyboard



Actions performed by the user



Additional information, notes and tips



Warnings whose observation is essential

2 Possible system bottlenecks

There are an enormous number of possible reasons for why the performance of a system may be unsatisfactory. Frequently a system suffers from a number of bottlenecks, some of which disguise others which can only be detected when the initial bottleneck has been eliminated. The following factors may cause bottlenecks:

- Application programs

The system may be slowed, for example, by the fact that tables are locked (wait state in masks) or that application data is gathered in server processes.
- Hardware:
 - Processors
 - Main memory
 - Hard disks
- Operating system:
 - System tables
 - File systems
- Other factors which are not discussed in this manual:
 - Remote data transfer connections (LAN, WAN)
 - Terminal connectors
 - BUS systems (system bus, I/O bus)

If one system component is overloaded, this has repercussions for the remaining components. Similarly, if one component is modified, this influences the behavior of the other components. For example, if the throughput of one component is increased, a bottleneck may suddenly occur at another component. For this reason, you should always analyze the system in its entirety before and after each tuning operation. You have obtained the optimum result when you have achieved balanced usage of all system resources.

2.1 Characteristics of the storage media

Computers process large volumes of electronic data which must be stored and made available. To this end, computer technology makes use of a number of storage media. These differ in capacity, access speed, dependence on power supply and cost. Figure 1 provides an overview of the most important storage media which are used in RM computers.

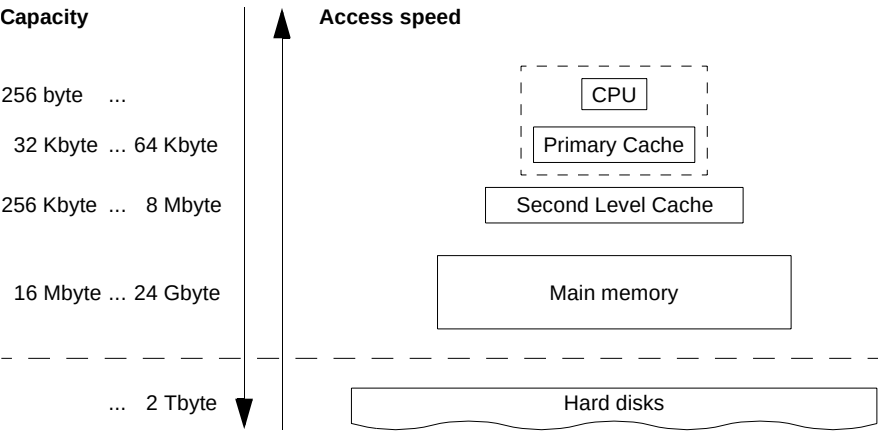


Figure 1: Characteristics of the various storage media

The capacities of the various storage media are approximately inversely proportional to their access speed. The CPU (Central Processing Unit) possesses the fastest storage, consisting of 32 8-byte registers. The CPU also possesses a cache memory, the so-called Primary Cache. The CPU can access this cache memory just as quickly as it can access its internal registers.

Somewhat slower than these two types of memory is an external cache memory: the so-called Second Level Cache (SLC). However, this memory is able to store up to 8 Mbytes of data. Then come the main memory (RAM = Random Access Memory) and the hard disk. Hard disks have capacities of up to several terabytes but are relatively slow when compared to the speed of the processor.

An important characteristic of these storage and memory media is their dependence on electricity. Some media provide only “temporary” (volatile) data storage (for example, the CPU registers and the cache memory). As a result, the data is lost when the computer is switched off. For this reason, the computer must also possess equipment for “permanent” (non-volatile) data storage (for example, hard disks).

2.2 Processors

Bottlenecks may occur at processors if a large number of processes or if processor-intensive processes are active in the computer. One indication of this is the presence of a large number of executable processes in the processor queue.

2.3 Main memory

It would be ideal if all the programs which you want to start and all the data they require could be fully loaded into the computer’s main memory. In practice, however, this is not possible, even in computers which have a relatively large main memory. For this reason, the operating system has the task of administering the available resources as skilfully as possible. To do this it proceeds as follows:

- Data/commands are not loaded into main memory until they are needed for processing.
- Data/commands which are used frequently are left in main memory for as long as possible.
- Sometimes space is created by removing infrequently used data and commands from main memory.
- Data/commands which are no longer needed are stored in “permanent” storage.

The smallest executable program unit is termed a process. The operating method outlined above allows UNIX to execute processes with a memory requirement of up to 1 Tbytes even though the main memory of the computer may be significantly smaller than this. The same is true of data: a program can process more data than can be stored in the computer’s main memory.

Paging and swapping

The entire contents of main memory (with the exception of the commands of the kernel and some associated files) can be transferred to hard disk page by page. This is done whenever,

- the pages are no longer used
- the number of free pages in main memory is too small

In the second case, (i.e. when the main memory no longer possesses sufficient memory pages), the system starts a paging operation in which individual pages are transferred from main memory to hard disk. Prior to this point (i.e. as long as sufficient free pages are available) the main memory is of no relevance for the analysis of system performance.

If the paging operation does not create a sufficient number of free pages, the system starts a swapping operation. In this case, entire processes are transferred to hard disk. If one of these transferred processes is then reactivated, it has to be loaded again from disk. This takes time and system performance suffers considerably in consequence.

In short, when many or very large processes have to be active, bottlenecks may occur in main memory.

You have a number of options if you want to improve the performance of the system as a whole:

- **Modifying the hardware**

You may, for example, increase the size of main memory. In this case, please contact your local Siemens Nixdorf Informationssysteme AG provider or branch.

- **Optimizing resource management**

You may be able to influence memory management by modifying certain UNIX parameters. In this way you can adapt the performance of RM computers to special operating conditions, for example if the computer is used for special tasks only or has a very high workload.

- **Modifying the kernel parameters**

System sequences are dependent on so-called kernel parameters. For more information, refer to the chapter “Configuring the kernel” on page 163.

2.3.1 Main memory segments

The entire main memory is not available for application programs, since UNIX requires part of the main memory itself. More precisely, the main memory comprises several areas known also as segments:

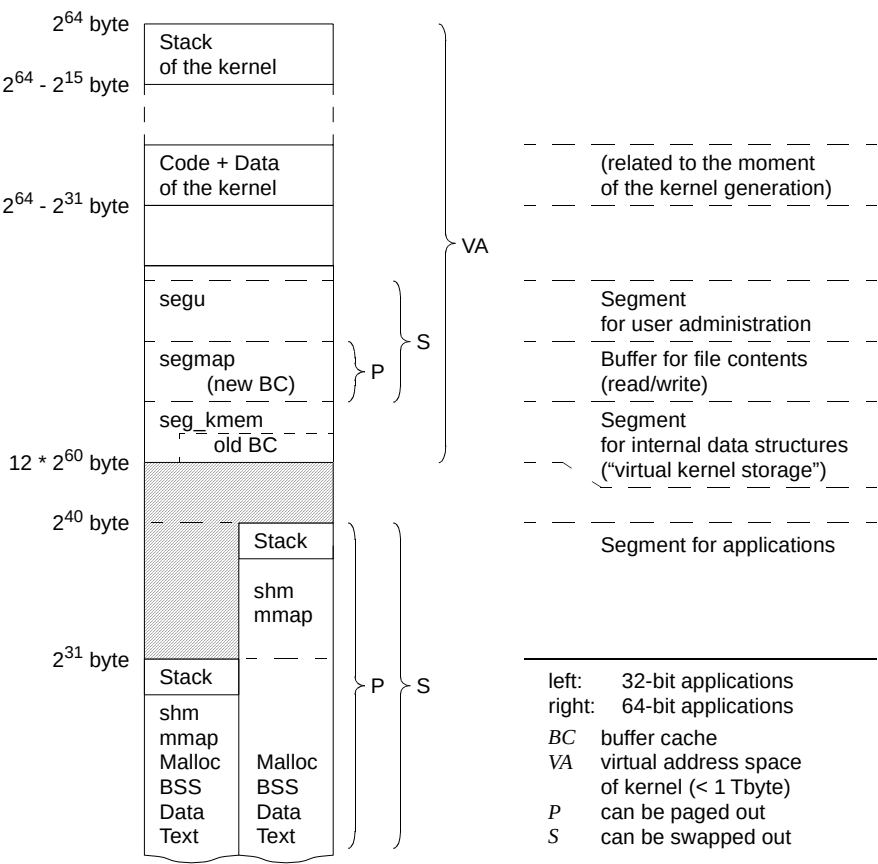


Figure 3: Areas in dynamically managed virtual main memory

Reliant UNIX 5.45 uses a 64-bit address space. The user address space is described by the small-code model. This model uses two universes:

- A 32-bit universe (bottom left-hand side of the figure above)

This contains all directly accessible statically linked binaries containing text and data in addition to all commonly accessible libraries. The address space in this case is 2 Gbytes.

- A 64-bit universe (bottom right-hand side of the figure above)

This contains, for example, the stack, mapped files and shared memory. The address space for these memory segments is 1 Tbyte. The heap area assigned using the *sbrk(2)* system call is located within the 2-Gbyte area behind the segment with the process data.

The process text segment starts at address *0x0000 0000 0040 0000* and is followed by the process data segment. The commonly accessible process libraries are mapped to the *0x0000 0000 0800 0000* address. The area between the process data segment and the commonly accessible libraries is used by the *brk(2)* system call. The process stack starts at address *0x0000 00ff ffff 0000* and expands downwards. The entire area between the stack and the commonly accessible libraries can be used for shared memory and memory mapped file segments. The 32-bit process address space behaves on a 64-bit system exactly the same as it does on a 32-bit system.

The size of the segments is determined by a number of kernel parameters:

- direct: by *KV_SIZE*, *NPROC* and *SM_SIZE*
- indirect: by other kernel parameters (see below)

If very many processes are active in your system or if it possesses a large number of active file systems (more than 30), the kernel must be correctly configured if the operating system is to run optimally.

2.3.2 *seg_kmem* segment

The segment *seg_kmem* is reserved for system-internal memory requirements. The size of this segment is defined by the kernel parameter *KV_SIZE*. The kernel parameter does not therefore specify how much memory is actually used but simply the maximum that is available for use. If necessary, the value of *KV_SIZE* may be increased on system start in order to ensure that the segment is large enough:

- First of all the system calculates the minimum value required for *KV_SIZE*.
- The minimum value is then compared with the value which is set in the file */etc/conf/cf.d/mtune* or */etc/conf/cf.d/stune*.
- The larger of the two values is used.

During the automatic calculation of the value of *KV_SIZE*, a number of kernel parameters are evaluated: *NPROC*, *BUFHWM*, *NBUF*, *PAGEIOPOOL_HWM*, *RBUFPOOL_HWM*, *NINODE* or *UFSNINODE* and *NDQUOT*.

You can calculate an approximate value for *KV_SIZE* using the following formulas (all values specified in Kbytes):

$$\begin{aligned} KV_SIZE &= kv_param \\ &+ kv_hwconf \\ kv_param &= NPROC \quad * \quad 16 \text{ Kbytes} \quad (\text{Contributions of kernel param.}) \\ &+ BUFHWM \quad * \quad 16 \text{ Kbytes} \quad (\text{system-internal buffer pool}) \\ &+ UFSNINODE \quad * \quad 0.8 \text{ Kbytes} \quad (* 2 \text{ in the case of a VERITAS fs}) \\ &+ \quad \quad \quad 2048 \text{ Kbytes} \\ kv_hwconf &= contr \quad * \quad 2048 \text{ Kbytes} \quad (\text{Contributions of hardware}) \\ &+ cpu \quad * \quad 256 \text{ Kbytes} \quad (\text{configuration}) \\ &+ physmem \quad * \quad 1536 \text{ Kbytes} \end{aligned}$$

where:

- contr* number of controllers
- cpu* number of processors (in RM600 E: 4 per CPU board)
- physmem* size of main memory (in units of 100 Mbytes)

The default kernel parameter values produce an address space requirement of at least 80 Mbytes for *kv_param* alone when the product is used on an RM600.

Old buffer cache

The *seg_kmem* segment contains a special area: the “old” buffer cache. The buffer cache is used for file system management (inodes, indirect blocks and file system management blocks). It does not store user data from files. This is the task of the *segmap* segment (see section “segmap segment” on page 14). In the literature, this segment is also known as the “new” buffer cache. However, in this manual the term buffer cache is used exclusively to refer to the old buffer cache.

You can configure the buffer cache using the kernel parameters *BUFHWM*, *NBUF* and *NHBUF*:

<i>BUFHWM</i>	specifies the size of the buffer cache in units of 16 Kbytes (see page 185). On system start, a check is performed to determine whether the size of the buffer cache lies within certain limit values. If necessary, the size of the buffer cache may be adapted as appropriate: – at least 1% of the physical main memory (no downsizing in the case of large systems) – maximum 5% of the physical main memory (no upsizing in the case of small systems) – maximum 8 Mbytes (if the kernel parameter <i>BUFHWM</i> <= 512)
<i>NBUF</i>	specifies the number of buffer heads which are definitively reserved for data buffer management on system start (see page 186).
<i>NHBUF</i>	specifies the number of hash entries which can be used to search for a buffer via device and block numbers (see page 187).

You can output the current values of the kernel parameters as described in the section “Determining the values of kernel parameters” on page 168.

2.3.3 *segu* segment

The *segu* segment contains kernel management data for the individual processes. The size of the segment is *NPROC* * 32 Kbytes (virtual) or *NPROC* * 16 Kbytes (physical). Since the default value of *NPROC* is calculated from *MAXUSERS* the size of *segu* is also indirectly dependent on *MAXUSERS*. With the default value of *MAXUSERS* = 512 the size of *segu* is approximately 129 Mbytes (virtual) or 65 Mbytes (physical)¹.

¹ The physical limit of 16 Kbytes or 65 Mbytes applies only for existing processes. Only the virtual address space is allocated in the whole size, e.g. for all *NPROC* processes.

2.3.4 segmap segment

In the literature, the *segmap* segment is also referred to as the “new” buffer cache. However, in this manual the term buffer cache is used exclusively to refer to the “old” buffer cache (see section “Old buffer cache”).

The memory segment corresponding to *segmap* is typically the largest. It comprises that part of the kernel address space which is not occupied by the two other segments. The *segmap* segment is used to store user data from files which processed by processes using the *write()*- and *read()* system calls. The size of the segment is determined by the *SM_SIZE* kernel parameter (as a percentage of the real size of the main memory). In view of the limitation of the virtual address space to 1 Gbyte, the maximum space of the *segmap* segment is as follows:

$$\text{segmap} = 1 \text{ Gbyte} - (\text{seg_kmem} + \text{segu})$$

If a process uses a *read()* system call to process a file, the data is first copied from the hard disk to the *segmap* segment from where it is copied to the data area of the user process. Conversely, in the event of a *write()* call, the data is first copied from the user process to the *segmap* segment and then from there to the hard disk. If multiple processes need to access the same data, this buffering procedure considerably reduces disk access operations. In addition, the file system is able to buffer more data than the user process requests and consequently reduce the need for disk accesses during subsequent read/write tasks.



The *segmap* segment grows dynamically at the expense of the remaining main memory which contains the user processes. This can result in an increased paging or swapping rate and consequently reduce performance.

User processes which map files to their own address space by means of the *mmap()* system calls achieve higher file input/output throughput than processes which copy data with *read()* or *write()*. The advantages are:

- During the transfer of so-called “memory mapped files”, no blocks have to be copied from the *segmap* segment to the buffer of the user application.
- Access to “memory mapped files” requires no system calls.

The above-mentioned advantage that multiple accesses to the same file require no additional disk input/output operations is also present when *mmap* system calls are used.

2.3.5 Transferring main memory pages

Regular file accesses (input and output of files) are performed via the *segmap* segment. This segment is subject to the operating system's paging mechanism. This means that under certain circumstances the file access buffer may be paged out. In contrast, input/output operations for file management (e. g. inodes) are performed via the buffer cache and this is not subject to the paging mechanism

If the amount of free memory falls below a certain limit value the kernel processes *pageout* and/or *sched* are activated. If *freemem* falls below the value of *LOTSFREE* the *pageout* daemon is called. This continues to free main memory pages until *freemem* again reaches the value of *LOTSFREE*. The value of *LOTSFREE* is defined in the file */usr/include/sys/vmparam.h*.

You can also use the kernel parameter *PAGEOUT_DELTA* to influence the behavior of the daemon (see page 191). This parameter defines the proportion of the paging address space which has to be additionally freed. This proportion is calculated as follows: $(PAGEOUT_DELTA / 100) * \text{size of the available paging address space}$.

The kernel parameter *GPGSLO* specifies when swapping begins (see page 191). The value of this parameter is defined using the file */etc/conf/cf.d/mtune* or */etc/conf/cf.d/stune*. As this process is performed largely automatically, *GPGSLO* has a value of -1. Refer to the chapter "Configuring the kernel" on page 163 for information on finding or modifying the current value of *GPGSLO*.

2.3.6 Page cache

Most input/output operations are performed via the page cache, for example loading programs, file input and output, paging and swapping. Paging and swapping - or, more precisely, the processes *pageout* and *sched* - ensure the recency of the page cache, i.e. if there is little free space left in main memory, then only those pages that have been used or modified sufficiently recently are buffered.

The page cache contains data belonging to user processes as well as data belonging to the kernel. User data may belong to multiple processes simultaneously (shared pages).

2.3.7 Shared memory alignment

The operating system requires main memory pages both for user data and for managing this data (e.g. address translation tables). This management data, also known as overhead, requires approximately 1 thousandth of the memory occupied by the process itself. The volume of data may become quite large. If, for example, a thousand processes occupy a shared memory of 10 Gbytes on average, the overhead will also amount to a total of 10 Gbytes.

This explosion of the memory space can be prevented by aligning the shared memory: The processes involved are assigned the same address space whereby the start address and size of the area must be a multiple of 64 Mbytes. In this case, the management data must only be generated once. In the above example, the overhead then only amounts to 10 Mbytes instead of 10 Gbytes.

You can use the *sysmips* system call to align shared memory segments. This command supplies the size and start address of the segment:

```
#include <sys/sysmips>
size_t sysmips(int SHMADDR_ALIGN, size_t shmsize, 0, 0);
```

The system call can be used in both 32-bit and 64-bit applications, but does not directly influence the *shmget*(2) and *shmat*(2) system calls.

2.3.8 Big pages

A virtual address from the program is used very time an application accesses the memory under Reliant UNIX. Before the CPU can access the memory, the virtual address must be converted to a physical address. To improve performance, hardware support is provided in the processor, namely the TLB (Translation Lookaside Buffer). The TLB is a cache for address translations. On the one hand, the TLB can be accessed extremely quickly but, on the other hand, the TLB can only store a relatively small number of address translations, generally the last ones used. Thus, to ensure effective caching of address translations, the entire address space is divided into blocks, known as main memory pages. All main memory pages (with the exception of big pages) have a fixed size of 16 Kbytes.

To minimize the time required for loading TLB entries (also known as overhead), several types of big page have been developed:

- Big page for shared memory data (16 Mbytes)
- Big page for user text (4 – 32 Mbytes)
- Big page for page tables (64 Kbytes, 256 Kbytes or 1 Mbyte)

A big page refers to a large contiguous memory area. By using big pages, you can reduce the number of dynamically required TLB entries. The system contains only one big page.

Big page for shared memory data

By default, a memory page for the shared memory segment is the same size as a “normal” main memory page, i.e. 16 Kbytes. To minimize the system overhead required for filling the TLB, shared memory segments use 16-Mbyte big pages. Only one TLB entry is required to map a 16-Mbyte page. Shared memory big pages are locked in the main memory and are not swapped out with paging. Thus, there are no paging and swapping activities in relation to these pages.

By default, the 16-Mbyte shared memory pages are assigned to all memory boards in accordance with the round robin algorithm. However, you can force the system kernel to only assign big pages to boards on which the accessing process is running. The system kernel then tries to assign the required pages to the desired board. If this is not possible, it assigns the pages to other boards on which the free pages are available. The size of the physical main memory to be used for shared memory big pages must be reserved in advance when booting the system.

The interfaces of the *shmget(2)* and *shmat(2)* system calls have not been changed. In order to achieve maximum optimization, the size and start address of the shared memory data segment must be aligned to the value returned by the *sysmips* system call (see page 16).

For this concept to work, a relatively large area of main memory must be reserved. It should therefore be restricted to cases where it is really necessary.

By default, big pages are therefore not used for shared memory data. To enable this function, specify the *segshm=1* option in either the Boot2 command line or in the */etc/default/boot* file when booting. Using the *segshm_askmem=*n** variable, you can specify how large the memory to be preloaded should be, where *n* is specified in Mbytes and is a multiple of 64 Mbytes.

Sample `/etc/default/boot` file:

```
B00TP4=segshm=1
B00TP5=segshm_askmem=64
```

The Reliant UNIX system kernel distributes the main memory pages required for shared memory segment page tables among all participants of the shared memory data big page.

Big page for user text

A user text big page contains instructions from the most important program in the system, typically the database. Only one TLB entry is used to map the entire text page. This TLB entry must be assigned permanently to eliminate all text TLB mismatches and to consistently increase the number of TLB hits for other memory segments. Text big pages are locked in the main memory and are thus not swapped out with paging. Thus, there should be no paging and swapping activities in relation to these pages.

The size of the big page is calculated based on the size of the user text segment and must be a multiple of 4 Mbytes. A text segment cannot exceed 32 Mbytes. At present, only one file can be loaded using *preload*. For further information, see *preload(8)*.

The user code in a big page is executed in a virtually mapped address space in cache mode.

Two additional optimizations have been carried out here:

- Copy the text big page to a big page for every CPU board.
RM600E: advantageous for utilizing the ccNUMA architecture
- Use non-coherent cache mode (if the hardware supports this mode).
Feasible for the RM600-xxx with CARIN-CPU and 200 ... 250 MHz
Not feasible for the RM600E

Big page for page tables

This type of big page is also useful for reducing the system kernel overhead required to load the TLB.

The need for this form of optimization arises if 64-bit applications create very large address spaces. This generally covers very large shared memory segments, for example for database systems. With the new 64-bit operating

system, shared memory segments can be up to 20 Gbytes in size depending on the configuration of the main memory. Compared with the previous 32-bit operating system, this figure is twenty times greater. Under these conditions, it is essential that big pages be used for shared memory page tables as the TLB would be heavily overloaded without this optimization.

Before launching into a detailed discussion of how to handle big pages for shared memory page tables, the relationship between page tables and the TLB is described below.

If the address translation required by an application to access the memory is currently missing from the TLB, the processor requests the system kernel to reload this entry using software resources. This process is known as a TLB mismatch. The system kernel uses an address translation table (page table), of which there is one for every process, when reloading.

Since the rate of TLB mismatches is very high for many applications, the handling of such mismatches is optimized as much as possible in the system kernel. This allows the system kernel to handle most TLB mismatches with less than fifteen instructions by reading an entry from the page table of the active process and entering it unseen in the TLB in accordance with the TLB mismatch address. This automatically replaces any arbitrary TLB entry. The selection of the entry to be replaced is random.

The system kernel must also use virtual addresses in the *KPTESEG* segment of the system kernel to access the page tables of the active process. TLB entries are also required for this type of access. When handling TLB mismatches, it is therefore possible that a second, nested TLB mismatch can occur (double mismatch). Handling a double TLB mismatch is considerably more time consuming for the system kernel than a single TLB mismatch. The number of TLB mismatches should therefore be kept to a minimum.

The best that this optimization can hope to achieve is to minimize the TLB entries necessary for page tables. Thus, for very large shared memory segments, we recommend that you follow a different approach than usual: when the system kernel accesses page tables (or TLB mismatches), the standard main memory pages (16 Kbytes) should not be used, rather the respective big pages. Big pages for page tables can be 64 Kbytes, 256 Kbytes or 1 Mbyte in size. There is a further advantage to using such big pages: once they are entered in the TLB, they cannot be replaced. The operating system can therefore automatically recognize when big pages are used since the number of TLB entries for page tables is very small.

When using big pages for shared memory page tables, it is essential that shared memory segments be aligned. The alignment requirements are considerably greater than for unoptimized shared memory mode (see below). Depending on the achievable alignment, there are different values for *bigpt_size*. In general, it is recommended to aim for the highest possible value for *bigpt_size*. However, in practice you need to compromise between the attainable optimization and the existing main memory configuration. The following table lists the possible values for *bigpt_size* and the associated shared memory alignment:

bigpt_size	Alignment
64 Kbytes	512 Mbytes
256 Kbytes	2 Gbytes
1 Mbyte	8 Gbytes

As system administrator, you need to take the following measures if big pages are to be used for shared memory page tables:

- ▶ Determine the size of the big pages to be reserved for the shared memory page tables.

For every 16 Kbytes of a shared memory segment, you need 4 bytes for the page table. The calculated requirement must be rounded off to the size of the big page (see the *bigpt_size* parameter).
- ▶ Set the parameters in the `/etc/conf/pack.d/kernel/space.c` file.

bigpt_size: 64 Kbytes, 256 Kbytes, 1 Mbyte
bigpt_count: Number of big pages of size *bigpt_size* to be reserved at system startup for shared memory page tables (a value of 0 disables this optimization fully)
- ▶ Next, generate a new system kernel using *idbuild*.
- ▶ Align the shared memory segments.

Both the start address and the size of the shared memory segments must be fully divisible by the values specified above. You use the same mechanisms to align the shared memory segments as for aligning big pages for shared memory data.

2.3.9 RM600 E: Software support for ccNUMA

ccNUMA stands for “cache coherent non-uniform memory access”. “Non-uniform” means that all processes can access all memory spaces but their access times can vary depending on the “distance” between the processor issuing the query and the memory cell being queried. In other words, instead of providing a global memory to which all processes have equally short (or equally long - depending on your point of view) access times, NUMA systems distribute the memory over all CPU/memory boards.

A ccNUMA architecture was implemented in the processors of the RM600E series. We will describe a number of tuning measures below that should help to reduce access times to a minimum when reloading the cache.

Coherent local board access is always faster than coherent access to the memory of another CPU board. The fastest type of access is local, non-coherent access (see the section “Kernel text window” on page 22”). Local board access has a further advantage in that the load on the SP bus is reduced.

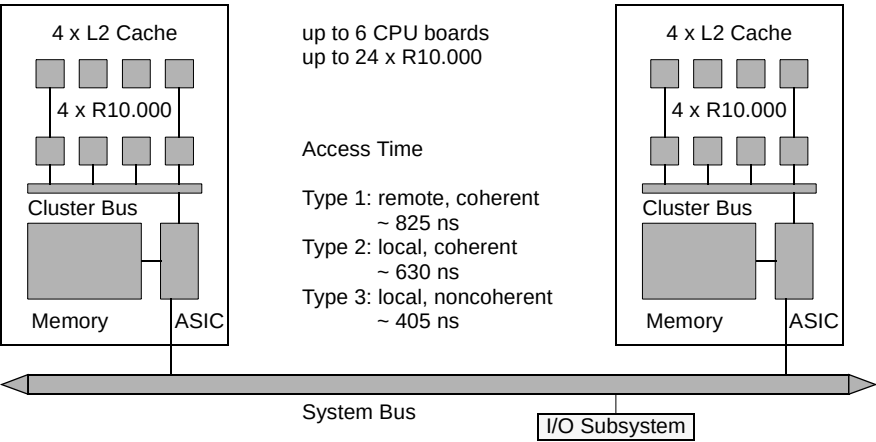


Figure 4: ccNUMA architecture in the RM600 E

CPU pools

A process always tries to satisfy its memory requirements locally on the board before a page is selected from the pool of pages on another board. The latter results if the local board pool has already been used.

Kernel text window

The system kernel text is duplicated on every CPU board in the system. In addition, it is mapped non-coherently. This allows mismatches in the secondary cache regarding system kernel instructions to be reloaded in the secondary cache at a speed of 405 ns. This feature is enabled by default. If required, you can disable it using the *r10k_kwin* variable in the */etc/conf/pack.d/kernel/space.c* file (1 = on, 0 = off).

User instruction page replication

This feature functions in the same way as the kernel text window for user applications. The aim is to always access the user instructions locally on the board. This is achieved by copying the user texts to every CPU board (on request). This, in turn, reduces the access time by approximately 20-30% when reloading a line from the secondary cache.

This feature is enabled by default. If required, you can disable it using the *r10k_cor_page* variable in the */etc/conf/pack.d/kernel/space.c* file (1 = on, 0 = off).

Reserving a shared memory page

By default, the shared memory is reserved page-by-page for all memory boards in accordance with the round robin algorithm. This guarantees that the shared memory pool can be accessed with the same speed from different boards. This is particularly important if there are several parallel accesses.

This reservation strategy can be selected using the *shmcntl* prefix command:

shmcntl -g *command* (round robin reservation of shared memory)

shmcntl -l *command* (local board reservation of shared memory)

It makes sense to opt for local board access to the shared memory pool when all processes involved are running on a single CPU board.

Example:

A RM600 E computer with two CPU boards has the following configuration (*autoconf -l*):

```
Cpu-Board0 in slot 2 ABIC-rev 8
  Logical CPU id 00 -- Physical id 00 agentno 0 PRId-Impl 9 PRId-Rev 2.6
  Logical CPU id 01 -- Physical id 01 agentno 1 PRId-Impl 9 PRId-Rev 2.5
  Logical CPU id 02 -- Physical id 02 agentno 2 PRId-Impl 9 PRId-Rev 2.5
  Logical CPU id 03 -- Physical id 03 agentno 3 PRId-Impl 9 PRId-Rev 2.6
  Onboard Memory: base 0x0 size 128 MB (2x64 MB)
Cpu-Board1 in slot 6 ABIC-rev 8
  Logical CPU id 04 -- Physical id 04 agentno 0 PRId-Impl 9 PRId-Rev 2.6
  Logical CPU id 05 -- Physical id 05 agentno 2 PRId-Impl 9 PRId-Rev 2.6
  Logical CPU id 06 -- Physical id 06 agentno 3 PRId-Impl 9 PRId-Rev 2.6
  Onboard Memory: base 0x8000000 size 64 MB (1x64 MB)
```

- Determine the number of processor groups in the system:

```
# cpumem -N
2
```

- Determine the CPU numbers, belonging to each processor group:

```
# cpumem -c 0
00 01 02 03
# cpumem -c 1
04 05 06
```

- Determine the memory belonging to each CPU group:

```
# cpumem -m 0
128
# cpumem -m 1
64
```

- Bind a shell to processor group 0

```
# mpcntl -b -c0 -c1 -c2 -c3 /sbin/sh
```

- Execute a process (*quickdd(1M)*) within this shell, which allocates his shared memory lokal from this processor group:

```
# shmctl -l quickdd if=/unix of=/dev/null bs=16k
523+1 records in,      8578621 bytes 6395.08kB/s
523+1 records out,     8578621 bytes 6494.23kB/s
```

For further information on the ccNUMA architecture, refer to the SNI Internet pages at the following address:

http://its.sni.de/lobs/its/sts/sinix/reports/addinfo/archiv/haphnum_en.htm

2.4 Hard disks and file systems

A considerable load is placed on hard disks and file systems when applications access disks intensively for input/output operations. This may, for example, be the case when database applications are run. The load and configuration of the hard disks may frequently be the cause of a bottleneck.

Disk throughput is also affected by the way data and programs are distributed across the disks. If all accesses are concentrated on one disk, this disk will be overloaded while a second disk may perform almost no work. If, in addition, the main memory is too small, the system may start paging or even swapping, thus further loading the disk with this additional input/output activity.

The size and location of the swap area on the disk are also of importance here.

Another way of improving performance in terms of disk utilization and load is to configure virtual disks (vdisk). The use of virtual disks makes it possible to distribute I/O work more effectively across the various disks (see section 6.3.1 on page 138).

VxFS file systems > 4 Gbytes

With a new installation, the */usr*, */var*, */opt*, */home* file systems are configured as VxFS file systems. If the VxFS file systems exceed 4 Gbytes, the following special conditions apply:

Number of inodes

The previous default value for this is:

Number of blocks / 4

If the VxFS file systems exceed 4 Gbytes (up to 16 Tbytes), this default value returns an extremely high number of inodes (approximately 250 million for every 1 Tbyte). This means that a correspondingly large volume of management information must be written with *mkfs*. However, a more serious concern is that a potentially necessary “full” *fsck* (*-I* option) may require more than four hours to check the consistency of a 1-Tbyte file system. Every inode block must be read regardless of whether or not it contains allocated inodes.

The *mkfs* command therefore restricts the number of inodes to approximately 1.05 million. This is therefore more or less the default value for a 4-Gbyte file system. The *mkfs* command takes about 7 minutes for a 1-Tbyte file system whereas the “full” *fsck* takes about 14 minutes.

The system administrator can still select a number of inodes as an *mkfs* option that differs from the default value. However, it should be clear that if the number of inodes greatly exceeds one million, extremely long *fsck* runtimes can result.

Problems that arise due to an excessively large number of inodes are described in the manual entitled “VERITAS File System (VxFS)” in the section on “mkfs(1M)”.

Allocation unit

The number of allocation units will be reduced by about half due to a modified algorithm in *mkfs*. This also saves on management information.

The system administrator can select the number of allocation units using *mkfs* options, either *nau=number* or *au size=number*. The effects on the *mkfs* or *fsck* runtimes are not as serious as with the number of inodes.

Intent log

The default size of the intent log is still 256 blocks. This number may be too small in cases of excessive load. In these cases, the system may hang, processes may “sleep”, particularly *fsflush*, and wait in the *vx_traninit()* routine for the transaction buffer.

This situation can only be avoided at present by increasing the size of the intent log with the *mkfs logs i ze=number* option. The previous upper limit of 1024 has been increased to 32768.

In the case of VxFS file systems that exceed 4 Gbytes and that must be checked with *fsck* when the system restarts after a *panic*, for example, you need to enter a number greater than 0 in the */etc/vfstab* file as the “level”. The swap environment is not yet installed at the time of level 0 so that the *fsck* main memory requirement for such file systems cannot be met.

VxFS file systems exceeding 4 Gbytes are generally configured as virtual disks of type *concatenated* or *striped*. Configuration as a virtual disk of type *striped* improves the performance because the disk load can be distributed. The VxFS allocation strategy is to distribute the directories over the allocation units and to allocate the data blocks of the files created in these directories locally to the allocation units in the directories. So, for example, if only one very large file is created in the root directory, only the blocks of allocation unit 0 are allocated initially. With a virtual disk of type *concatenated*, this means that I/Os are only initiated for the first disk. With a virtual disk of type *striped*, this I/O load can be distributed over several disks.

You should also note that a maximum of 99 pieces can be specified for every virtual disk configuration.

2.5 System tables and kernel parameters

The kernel manages internal tables in which it makes entries during operation. The size of these tables is defined using kernel parameters. The following table lists only those kernel parameters which specify the size of system tables (sizes for RM600).

Kernel parameter	Meaning	Min.	Max.
<i>NPROC</i>	Size of the process table	100	12308 188
<i>NCSIZE</i>	Size of the DNLC	100	4096 187
<i>UFSNINODE</i>	Size of the i-list (inode table)	100	46763 190
<i>MSGMNI</i>	Number of message queues	32	16384 194
<i>SEMMNS</i>	Number of semaphores	60	5000 195
<i>SHMMNI</i>	Number of shared memory segments	100	2000 196

Table 1: Kernel parameters which specify the size of system tables

The internal tables which are defined in the kernel must be sufficiently large for the application. If the limit is reached, the kernel issues corresponding error messages. The application may be aborted if necessary.

The system is supplied with the kernel parameters set to their default values and in most cases it is not necessary to modify these. Only in exceptional cases or in the event of special application profiles is it necessary to modify a parameter in order to improve performance or eliminate a bottleneck.

There are two reasons for modifying the preset values of the core parameters:

- The kernel is no longer operating efficiently since the “old” buffer cache is too small. This cache is located in the *seg_kmem* segment (see figure 3 on page 10) and is used for file system management (inodes, indirect blocks and file system management blocks).
- An application’s system calls fail because the resources specified by the kernel parameters are exhausted (e.g. semaphores or message queues).

In particular, interprocess communications (message queues, semaphores, shared memory) are configured using kernel parameters. If an application system makes intensive use of the interprocess communications mechanisms then it is essential to examine the message queue (see “Analyzing message

queues” on page 126), semaphores (see “Analyzing semaphores” on page 128) and shared memory (see “Analyzing shared memory” on page 130) loads when analyzing the overall system.



If you work with ORACLE® databases you will need to modify a number of kernel parameters. Please refer to the relevant “Installation and Configuration Guide”.

3 Tools for measuring performance

This chapter starts by listing those commands which can be used to measure performance. The most important commands are then described in detail.

3.1 Summary

You can use the following commands to perform measurements and analyses:

<i>autoconf</i> (8)	Output hardware configuration
<i>crash</i> (1M)	Check core dumps
<i>df</i> (1), <i>dfspace</i> (1)	Output file system allocations
<i>dispadm</i> (1M)	Display, set, check scheduling parameters
<i>dkconfig</i> (8)	Output I/O statistics for the virtual disks
<i>dkmap</i> (8)	Output disk allocations
<i>dkpart</i> (8)	Output disk partitions
<i>dkstat</i> (1M)	Output I/O statistics for all disks (the statistics relate to the period since the last system start or the last I/O counter reset)
<i>iotal</i> (1M)	Statistics monitor for I/O requests
<i>ipcs</i> (1)	Output the status of interprocess communications mechanisms (useful, among other things, for the output of information relating to active and shared main memory)
<i>mpcntl</i> (1M), <i>launchit</i> (1M)	Control of multiprocessor-specific utilities
<i>mpstat</i> (1M)	Output detailed statistics about CPUs as well as paging and swapping activity
<i>pagesize</i> (1)	Output the size of a virtual page
<i>prf</i> (7)	Operating system profiler
<i>priocntl</i> (1)	Display, set, check scheduling parameters
<i>procso</i> (1M)	List process and kernel memory usage
<i>ps</i> (1)	Output process data (process status); <i>/usr/ucb/ps</i> is also of use
<i>sadc</i> (1M)	Query system activities and store in binary form

<i>sar</i> (1)	Measure system activity – CPU load – Memory load – Paging and swapping activity – Disk load (most important and versatile analysis) In order to query the activity counters <i>sar</i> calls <i>sadc</i> or uses files which have been created by <i>sadc</i>
<i>scstat</i> (8)	Output system call statistics
<i>simon</i> (1)	Graphic performance monitor, <i>SINIX/monitor</i>
<i>swap</i> (1M)	Display or modify swap segments
<i>sysdef</i> (1M)	Display current values of kernel parameters
<i>truss</i> (1)	Log system calls (evaluation requires a good knowledge of C)
<i>timex</i> (1)	Measure the real time, user time and system time of a process
<i>uptime</i> (1)	Display average system load of the local computer
<i>vmstat</i> (1M)	Output statistics for processes, virtual mem. and CPU activity

Commands for the analysis of computer network performance

These commands are listed here for the sake of completeness only. They are described in detail in the manual “Network Administration”.

<i>arp</i> (1M)	Display and check address resolution
<i>etherstat</i> (1M)	Display network interface statistics Ethernet, Token Ring and FDDI controller (FDDI = Fiber Distributed Data Interface) interfaces are supported
<i>ifconfig</i> (1M)	Inspect and configure network interface parameters
<i>netstat</i> (1M)	Display number of incoming and outgoing network packets per controller
<i>nfsstat</i> (1)	Display statistics about global file systems (NFS = Network File System) and Remote Procedure Calls (RPC)
<i>ping</i> (1M)	Measure return times of network packets
<i>ruptime</i> (1)	Display average system load of the computers in the local network
<i>spray</i> (1M)	Measure mean packet throughputs between two computers
<i>tcpdump</i> (1)	Analyze data traffic in the network

3.2 sar(1)

The operating system uses a number of internal counters which are incremented whenever the system performs certain operations. Counters exist for:

- CPU load,
- Buffer allocation,
- I/O activity of disks and tape drives,
- Activity of TTY devices,
- Process switching activity and system calls,
- File accesses,
- Queue activity,
- Interprocess communications,
- Paging (in and out),
- Remote file sharing.

The mode of operation of the counters is described under *sar(1)*. The counters are periodically polled by the processes *sa1*, *sa2* and *sadc* (see *sar(1M)*). In addition, the directory */var/spool/cron/crontabs* contains a file called *sys* with the following entries:

```
0 * * * 0-6 /usr/lib/sa/sa1
20,40 8-17 * * 1-5 /usr/lib/sa/sa1
5 18 * * 1-5 /usr/lib/sa/sa2 -s 8:00 -e 18:01 -i 1200 -A
```

The shell script */usr/lib/sa/sa1* gathers all the data from the activity counters and stores it in the binary file */var/adm/sa/sa<dd>*. The shell script */usr/lib/sa/sa2* does practically the same thing. However, it generates a text file named */var/adm/sa/sar<dd>*. In both cases *dd* represents the current day.

The command *sar(1)* gathers the data accumulated by the operating system activity counters (*sar* = system activity reporter). It has the following syntax:

sar [*option*] [-**o** *file*] [-**P**[*number*] [*option*]] *interval* [*number*]

interval is the time between measurements in seconds, *number* is the number of measurements. Specifying the -**o** switch directs output to the binary file *file*.

The *sar* command also counts its own activity. Consequently the value of *interval* should be greater than 5 seconds in order to avoid excessive measurement distortion. As of an interval of only 10 seconds, measurement distortion caused by the simultaneous execution of *sar* is reduced to less than 1% and is therefore negligible. This order of interval is optimal for measurements over short periods (e.g. two hours). For long periods of measurement (e.g. a full day) an interval of 60 seconds is advisable.

The table below lists the options which can be specified with the *sar* command. For more details on the corresponding output, refer to the specified page in the chapter “Evaluating load measurements” on page 77. If no option is specified then the command behaves as if the *-u* switch had been specified.

Option		Displayed information
-a	122	Resolution of pathnames
-A		All data (equivalent to specifying all other options)
-b	97	System buffer load in main memory (“old” buffer cache)
-c	125	Number of system calls
-d	116	Activity of block devices (e.g. hard disks and tape drives)
-g	100	Page-out activity
-K	98	Kernel memory allocation
-l		Number of locks and resulting wait times
-m	128	Message and semaphore activity
-p	102	Page-in activity
-P	84	CPU load per processor This option is only available on multi-processor computers (possible on RM400 as of model 730 as well as on all RM600 models). <i>-P</i> may be specified in conjunction with the options <i>-b</i> , <i>-c</i> , <i>-l</i> , <i>-m</i> , <i>-p</i> , <i>-w</i> and <i>-y</i> .
-q	85	Average queue length and load (run queue)
-r	93	Free memory pages and free disk blocks in swap area
-u	115	CPU load (user mode, system mode, I/O wait state)
-v	95	Number of entries and size of system tables (process, inode, file and lock tables)
-w	105	Swapping and switching activity (in and out)
-x	123	Activity of UFS file systems
-y		Terminal activity

Table 2: Options of the sar(1) command

Example output for `sar -u 10 8`:

ReliantUNIX-Y wait 5.45 A0061 RM600 05/26/99						
10:13:59	%usr	%sys	%wio	%idle	int/s	intdef/s
10:14:09	11	11	44	34	784	0
10:14:19	13	12	31	45	776	0
10:14:30	12	11	35	42	792	0
10:14:40	11	9	42	38	798	0
10:14:50	12	7	40	40	803	0
10:15:00	17	8	44	32	805	0
10:15:10	3	63	17	17	781	0
10:15:20	2	23	38	37	1030	0
Average	10	18	36	36	821	0

The following sequence of analyses is useful when detecting performance bottlenecks:

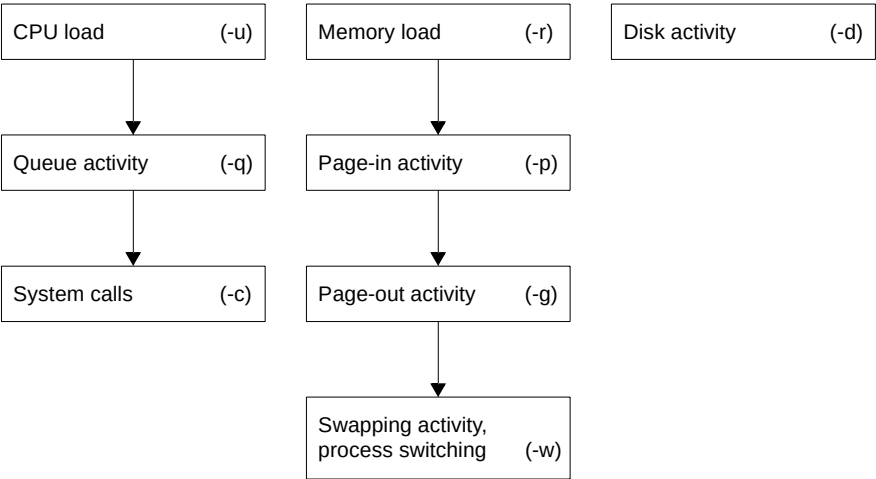


Figure 5: Sequence of sar evaluations

You can also display system activity in graphical form by means of the commands `xload` (see page 58) or `simon(1)` (see page 60). However, these commands are only available if your terminal has graphics capabilities (e.g. an X terminal or a PC with X emulation).

3.3 sysadm(1M)

You can also use the SYSADM user interface to monitor system activity. To do this, proceed as follows

- ▶ Start SYSADM.

The main menu of SYSADM appears.

- ▶ Select the menu item *performance*.

The menu *Monitoring System Activity and Workload* now appears.

- ▶ Select the menu item *sar*.

The menu *System Activity Report* is displayed:

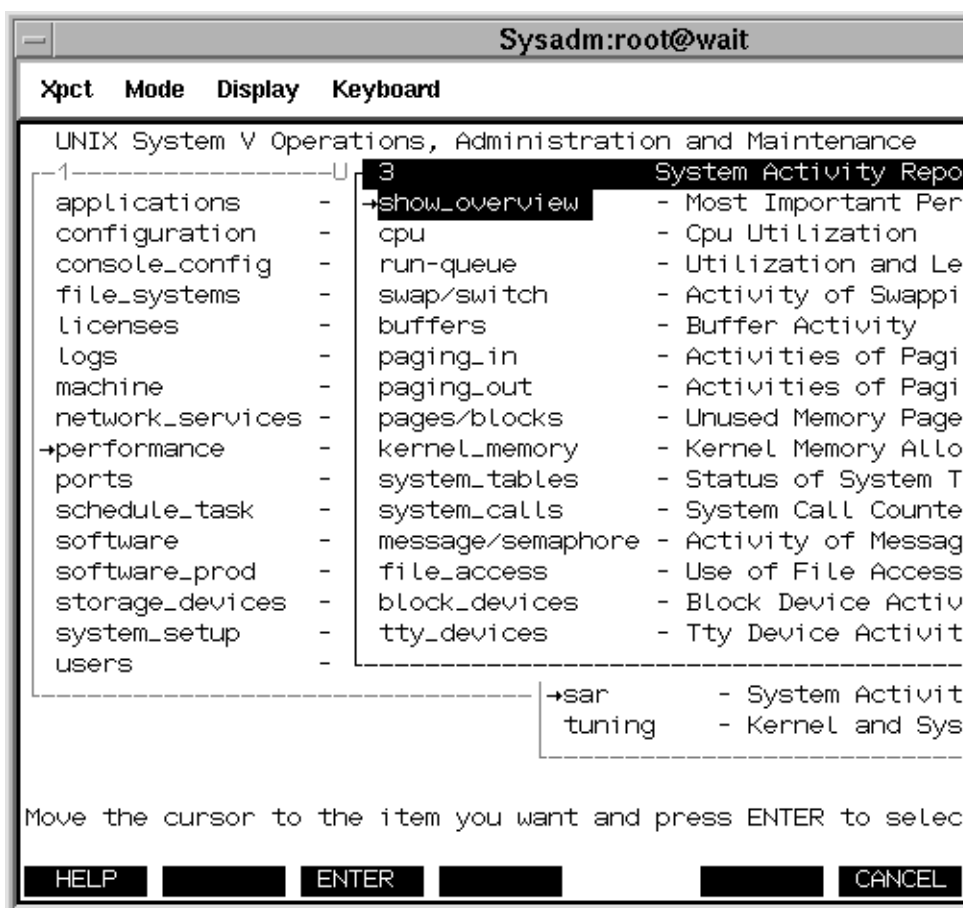


Figure 6: Menu SYSADM – performance – sar

The individual menu items correspond to the following *sar* options

Menu item	option		Displayed information
<i>show_overview</i>			The most important system activities
<i>cpu</i>	-u	115	CPU utilization (user mode, system mode, I/O wait state)
<i>run_queue</i>	-q	85	Average queue length and utilization (run queue)
<i>swap/switch</i>	-w	105	Swapping and switching activity (in and out)
<i>buffer</i>	-b	97	System buffer load in main memory ("old" buffer cache)
<i>paging_in</i>	-p	102	Page-in activity
<i>paging_out</i>	-g	100	Page-out activity
<i>pages/block</i>	-r	93	Free memory pages and free disk blocks in swap segment
<i>kernel_memory</i>	-K	98	Kernel memory allocation
<i>system_tables</i>	-v	95	Number of entries and size of system tables (process, inode, file and lock tables)
<i>system_calls</i>	-c	125	Number of system calls
<i>message/semaphore</i>	-m	128	Message activity
		130	Semaphore activity
<i>file_access</i>	-a	122	Use of system routines for file access
<i>Block_devices</i>	-d	116	Block device activity (e.g. hard disks and tape drives)
<i>tty_devices</i>	-y		Terminal activity

Table 3: *sar* options in the SYSADM menu item *System Activity Report*

Example output for the menu item *show_overview*:

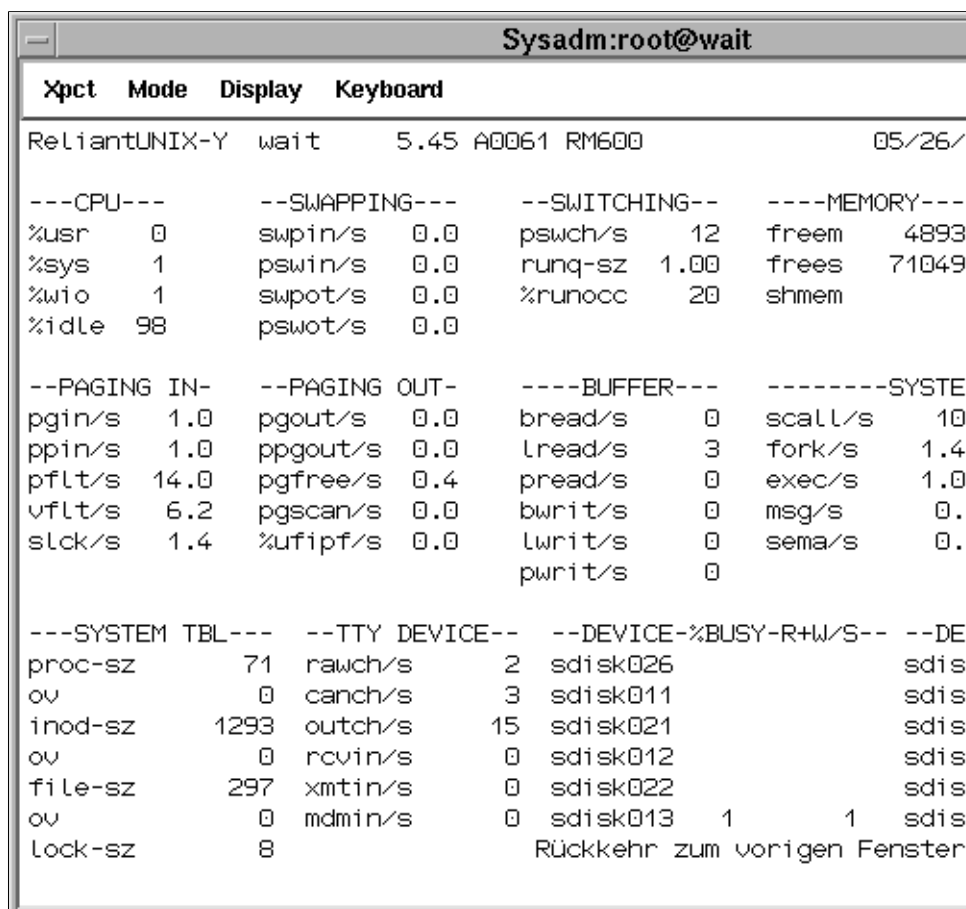


Figure 7: Displaying the most important system activities using SYSADM

The specifications correspond to the *sar* commands which are listed in the following table. For further information on the individual columns please refer to the specified page in this manual.

CPU	SWAPPING	SWITCHING	MEMORY
<i>sar -u</i> 115	<i>sar -w</i> 105	<i>sar -w</i> 105 <i>sar -q</i> 85	<i>sar -r</i> 93

FILE ACCESS	PAGING IN	PAGING OUT	BUFFER
<i>sar -a</i> 122	<i>sar -p</i> 102	<i>sar -g</i> 100	<i>sar -b</i> 97
SYSTEM CALLS	SYSTEM TABLE	TTY DEVICE	DEVICE
<i>sar -c</i> 125	<i>sar -v</i> 95	<i>sar -y</i>	<i>sar -d</i> 116
<i>sar -m</i> 128			

3.4 dkstat(1M)

The *dkstat(1M)* command provides comprehensive disk statistics. It has the following syntax:

dkstat *option* [*drive*]

You can specify *drive* in order to select individual hard disks. If this parameter is not specified then the five most active disks are displayed by default. If you do not specify an option, the command outputs a usage message. The table below lists the most important options which can be specified for the *dkstat* command.

Option	Displayed information
<i>-b</i>	Number of sectors (512 bytes) per I/O job
<i>-g</i>	Number of read/write jobs and the number of bytes transferred
<i>-p</i>	I/O distribution across the available partitions A section of the hard disk is known as a partition.
<i>-z</i>	Delete I/O statistics This option must always be specified in a separate line.

dkstat logs statistics since the last system start or since the last time the statistics were deleted (using the option *dkstat -z*).

Example for *dkstat -b*:

Block size profile (sectors per I/O)					
range	0s011	0s021	0s022	0s023	0s024
0	0	0	0	0	0
1	0	95	0	105	261
2	32	80	0	19	100
3	0	59	0	11	54
4	20	72	0	14	157
5	0	35	0	4	29
.	.	.	.	.	.
.	.	.	.	.	.

76 - 83	0	0	0	0	0
84 - 91	0	0	0	0	0
92 - 99	0	33	0	28	68
100 - 107	0	0	0	0	0
108 - 115	0	0	0	0	3
116 - 123	0	0	0	0	0
124 - 131	0	288	0	282	690
total	180	6588	6728	5365	6564
average:	14.4	33.4	32.0	35.6	37.9

You can use *dkstat -b* to determine whether more small (512 bytes) or more large I/O jobs (16 Kbytes) are sent. If, for example, many small I/O jobs are being executed and no large ones, this may be due to inefficient programming of the application.

Example for *dkstat -g*:

Totals					
counts	0s011	0s021	0s022	0s023	0s024
read	0	360	145	424	304
write	1	1305	1825	1581	1173
byte	8192	27731968	32276480	32432640	20852736

Example for *dkstat -p*:

Partition use distribution (I/Os by partition)					
range	0s011	0s021	0s022	0s023	0s024
0	245	0	0	0	0
1	0	0	0	0	0
2	3	0	0	0	0
3	74	0	0	0	0
4	64	0	0	0	0
5	9	0	0	0	0
6	0	0	0	0	0
7	0	0	0	0	0
8	0	9805	11075	8732	10363
total	395	9805	11075	8732	10363
average:	1.3	8.0	8.0	8.0	8.0

This listing makes it very easy to check, for example, that all I/O activity is concentrated in a single partition. If one partition is excessively busy then you should use “striped disks” to distribute I/O across multiple physical disks (see page 139).

3.5 mpstat(1M)

The *mpstat(1M)* command provides information about processes, virtual memory, exception handling and CPU activity. It collates important information from the *ps* command and various *sar* outputs. The syntax is as follows:

```
/usr/ucb/mpstat [interval] [number]
```

In the first line, *mpstat* outputs a summary of system activity which has occurred since the last system restart. If *interval* is specified, the following lines contain summaries for the last *interval* seconds. If *number* is specified, the statistics are repeated *number* times.

Example output for */usr/ucb/mpstat 5 3*:

procs										memory										page				-----Cpu 0-7-----									
r	b	w	avm	fre	pi	po	si	so	int	cint	fint	syc	cs	id	us	sy	io																
3	30	0	2192368	4200352	63	262	3	213	37	22	1	770	164	79	3	18	9																
					55	264	3	278	37	18	1	615	136	79	3	18	8																
					54	249	5	432	37	18	1	533	116	78	3	20	8																
					54	270	5	310	37	17	1	305	80	78	2	19	7																
					61	230	1	51	37	20	1	769	206	79	3	18	8																
					46	245	2	104	37	17	1	670	191	79	3	18	8																
					48	221	3	191	37	15	1	550	153	78	3	19	8																
					49	228	4	243	37	16	1	478	120	79	3	18	8																
0	30	0	2183696	4229424	0	0	0	0	70	0	0	3996	607	92	1	7	92																
					0	0	0	0	70	0	0	1889	297	93	0	6	93																
					0	0	0	0	70	0	0	2135	306	94	1	6	94																
					0	0	0	0	70	0	0	724	98	86	0	14	86																
					0	0	0	0	70	0	0	3810	596	89	0	11	89																
					0	0	0	0	68	0	0	2978	471	90	0	10	90																
					0	0	0	0	68	0	0	2941	462	93	1	6	93																
					0	0	0	0	70	0	0	2344	341	91	1	9	91																
procs										memory										page				-----Cpu 0-7-----									
r	b	w	avm	fre	pi	po	si	so	int	cint	fint	syc	cs	id	us	sy	io																
4	30	0	2189152	4231056	0	0	0	0	84	0	0	5004	758	90	1	9	90																
					0	0	0	0	84	0	0	2285	363	89	0	10	89																
					0	0	0	0	84	0	0	2719	368	89	1	10	89																
					0	0	0	0	84	0	1	946	124	86	0	13	86																
					0	0	0	0	84	2	0	4795	730	84	1	14	84																
					0	0	0	0	86	0	0	3867	619	92	0	8	92																
					0	0	0	0	84	1	0	3439	543	92	0	8	92																
					0	0	0	0	84	0	0	2752	415	94	0	6	94																

The columns have the following meaning:

- procs
- Number of processes in various states

r

In a run queue

b

Blocked because of a resource shortfall (I/O or paging)

w

Executable or briefly sleeping (< 20 seconds) but swapped out

memory

Use of virtual and real memory

avm

Active virtual pages of 1024 bytes

The pages belong to processes which have run in the last 20 seconds or are still running

fre

Size of the memory freelist in pages of 1024 bytes

page

Paging and swapping activity

pi Number of pages paged in per second

po Number of pages paged out per second

si Number of pages swapped in per second

so Number of pages swapped out per second

Cpu

CPU activity

int

Device interrupts processed per second

cint

Messages from one CPU to the other processed per second

fint

Number of floating point interrupts processed per second

sys

System calls per second

cs

Number of process switches per second

id

Percentage of time in which CPU is idle or in I/O wait state

us

Percentage of time in which CPU is in user mode

sy

Percentage of time in which CPU is in system mode

io

Percentage of time in which CPU is in I/O wait state

If a large number of processes are blocked (*b*) or are being swapped out (*w*) and have to wait before being executed, this may indicate a memory bottleneck. The size of the free memory *fre* is presumably relatively small but may vary greatly if some processes are scheduled for execution and others are swapped out to make room for processes which have to be swapped in.

If the values of *pi* and *po* are continuously high over an extended period, this means that pages are continuously being paged in and out. If at the same time the values of *fre* are too small this is a clear indication that the main memory is too small.

If *cint*, the interrupt rate between the CPUs, is very high, the TLB (Translation Lookaside Buffer) or the management of the command cache requires a lot of time. The TLB is an associative cache memory which is located in the CPU. It contains information for the mapping of virtual and physical memory addresses. The *cint* rate grows if processes are switched frequently between CPUs or if there is a high level of paging activity. High levels of paging activity occur when the *pageout* process is running and is continually searching for memory space as a result of a memory shortage. If *pageout* is extremely active for one CPU then there is an intensive exchange of messages between the other CPUs, informing them of the necessity of modifying their TLBs and caches.

The rate of floating point interrupts *fint* comprises both floating point errors and FPU ownership rights. If this rate is exceptionally high, this may indicate the presence of a deviant process which is experiencing problems with its *SIGPFE* signal handling.

If the rate of process switching *cs* is high, this indicates that there are a large number of active processes and, under certain circumstances, that the time slices in the dispatcher table are too short.

If there is an exceptionally high level of paging activity, then the I/O wait time *io* may greatly exceed the idle time of the CPU. In addition, I/O-intensive applications may increase the percentage value of I/O wait times if no other users want to use the CPU during the wait time.

For further information on the *mpstat* command output refer to the section “Swapping activity” on page 104.

3.6 vmstat(1M)

The *vmstat*(1M) command supplements the *mpstat*(1M) command described above. It combines statistics for the various CPUs and supplies detailed information about the paging mechanism. It has the following syntax:

/usr/ucb/vmstat [*option*] [*interval*] [*number*]

interval is the interval between measurements in seconds, *number* is the number of measurements.

The table below lists the options which can be specified for the *vmstat* command.

Option	Displayed information
-d	Output the disk assignments
-S	Output statistics as if no option had been specified Difference: The columns <i>si</i> and <i>so</i> are output in place of the columns <i>re</i> and <i>at</i> (i.e. number of pages swapped in and out per second, see page 41)
-f	Output the number of forks
-m	Output the number of physical pages currently being used by the operating system
-s	Output the number of paging and swapping operations

Example output for */usr/ucb/vmstat -d*:

```
disk assignments
d0 = ios0/sdisk011
d1 = ios0/sdisk110
d2 = ios0/sdisk012
d3 = ios0/sdisk111
d4 = ios0/sdisk112
d5 = ios0/sdisk060
d6 = ios0/sdisk114
.
.
.

procs          memory          page          cpu(s)
r  b  w    avm   fre    re  at   pi po fr sr    in  syc    cs us sy id io
1 28 0 2177456 4242896      3  17 424 1958 1982 6909    290 4694 1163  3 19 71
8
0s01: 0      0s11: 1      0s06: 0      0s10: 1      Total: 2
```

The columns have the following meaning:

- procs
- Number of processes in various states

r

In a run queue

b

Blocked because of a resource shortfall (I/O or paging)

w

Executable or briefly sleeping (< 20 seconds) but swapped out

memory

Use of virtual and real memory

avm

Active virtual pages of 1024 bytes

The pages belong to processes which have run in the last 20 seconds or are still running.

fre

Size of the memory freelist in pages of 1024 bytes

page

Paging and swapping activity

re

Pages reclaimed back per second

Number of pages which were written to a file or swap area and reclaimed back before the I/O operation was terminated

at

Linked pages per second

Linked pages are pages which were located in the memory freelist before being paged out to disk

pi

Number of pages paged in per second

po

Number of pages paged out per second

fr

Pages freed per second

sr

Pages checked by the *pageout* daemon per second

cpu(s)

Activity at each CPU

in

Device interrupts received per second

sys

System calls per second

cs

Number of process switches per second

us

Percentage of time in which CPU is in user mode

sy

Percentage of time in which CPU is in system mode

id

Percentage of time in which CPU is idle or in I/O wait state

io

Percentage of time in which CPU is in I/O wait state

The counters *re* and *at* are used for the following situation: Pages are often reclaimed back when the *pageout* daemon initiates an I/O operation in order to page a page out to disk and the process accesses the page before the I/O operation is completed.

The *fr* rate indicates the number of pages freed (“stolen”) by the *pageout* daemon. If this rate is very high and there is also a high level of disk activity, this may mean that the system is being slowed because it has to wait for disk accesses.

The *sr* rate indicates how busy the *pageout* daemon is. The time which is consumed by the daemon cannot be used by applications. As a result, a high level of daemon activity indicates that a memory extension is required.

The *-s* option causes *vmstat* to output the values of all the memory management counters which have been incremented since the last system start.

Example output for `/usr/ucb/vmstat -s`:

```

109618 swap ins
246543 swap outs
1753888 pages swapped in
148396272 pages swapped out
565185 total address trans. faults taken
2094886 page ins
10023819 page outs
34667472 pages paged in
160247824 pages paged out
1731798 total reclaims
1428117 reclaims from free list
12714 intransit blocking page faults
9197024 swap text pages found in free list
13635312 inode text pages found in free list
434341888 pages examined by the clock daemon
1289 revolutions of the clock hand
161851392 pages freed by the clock daemon
95073035 cpu context switches
23685456 device interrupts
7603111 traps
384001687 system calls

```

If you use this command in conjunction with the `/usr/ucb/uptime(1)` command it is very easy to determine whether a system has paged out a large number of main memory pages since the last system start. If this is the case, the system may not possess sufficient memory.

For further information on *vmstat* command output, refer to the section “Paging activity” on page 100.

3.7 scstat(8)

The *scstat(8)* command provides extremely detailed information about system calls. It has the following syntax:

scstat [*option*] [-o *file*] [*interval*] [*number*]

interval specifies the period between measurements in seconds, *number* is the number of measurements. Specifying the -o option directs output to the file *file*. The table below lists the options which can be specified for the *scstat* command.

Option	Displayed information
-a	Activate monitoring of system calls
-p <i>pid</i>	Activate monitoring of system calls for the <i>pid</i> process
-d	Deactivate monitoring of system calls
-c	List number of system calls per CPU By default the system calls are listed for all processors jointly
-s	List the number of system calls cumulatively
-t	Sort and list system call information System calls are sorted on percentage duration
-z	Delete system call monitoring statistics



The options -a, -p and -d are mutually exclusive. *root* permission is required for the use of these options.

Before commencing measurement you must first activate the mechanism for system call statistics (-a option). It is also advisable to delete old or current values before commencing measurement (-z option). When measurement is completed, you can delete the *scstat* statistics using the *scstat -d* command.

Example output for *scstat -c 30 5*:

```
Total calls/sec 1548. By (CPU):
(0)  826 (1)  721

CPU(0)____calls/sec_%time  CPU(1)____calls/sec_%time
close          121    0.5    xstat          104   11.0
xstat          120   10.6    close          104    1.0
open           91    3.9     open           78    4.5
write          63    0.5     read           54    0.9
read           61    1.2     fork            52    9.7
fork           60   11.2     fcntl           52    1.2
signal         60    0.6     signal          52    0.3
fcntl          60    0.6     write           51    1.0
brk            31    0.5     ioctl           27    0.3
execve         30   50.9     execve          26   50.3
_exit          30   14.8     waitsys         26    1.2
waitsys        30    1.2     getpid          26    0.2
ioctl          30    0.6     _exit           25   17.4
getpid         30    0.2     brk             25    0.5
lxstat         3    0.3     alarm           3    0.0
unlink         < 1    2.4     sigaction        3    0.0
lseek         < 1    0.0     sigprocmask      2    0.0
getdents      < 1    0.0     sigsuspend       1    0.2
mmap           < 1    0.0     lseek            1    0.0
access        < 1    0.0     context          1    0.0
context       < 1    0.0     access          < 1    0.0
fxstat        < 1    0.0     sysmips         < 1    0.0
                                poll            < 1    0.0
                                pgrpsys        < 1    0.0
                                hrtsys         < 1    0.0
```

In the example above there were a total of 1548 system calls per second. Of these, 826 system calls per second related to CPU(0) and 721 system calls per second to CPU(1). *scstat* then displays the breakdown of these system calls for each CPU. It specifies how often each system call was activated and the percentage of CPU time which this type of call consumed.

This makes it clear that some system calls are more “expensive” than others. For instance, in the above example 63 *write* system calls per second accounted for 0.5% of the time whereas 30 *execve* system calls claimed 50.9% of CPU time. A *write* system call is therefore “cheap” compared to an *execve* system call.

The interpretation of system call statistics is usually a difficult operation. It does, however, give application programmers the opportunity to detect weaknesses in their programming (for example: character-by-character *read()* calls when reading from a large file).

scstat can also be used to obtain the system call statistics of a single process. This is necessary for the analysis of applications. To do this you must specify the *scstat* option *-p* and the required process ID (PID). The *scstat* statistics then relate to this process only.

3.8 iotal(1M)

iotal continuously prints the service times (in micro-seconds) of I/O requests as they are completed for a single device. Typically, this information is used to produce graphs describing the performance characteristics of an I/O subsystem or to follow the device usage pattern of an application.

Any number of *iotal* commands can be simultaneously active on a system for any combination of traceable devices. This includes several invocations of *iotal* tracing the same device.

The command has the following synopsis:

iotal [*option*] [-m *num*] [*device*]

The table below lists some options which can be specified for the *iotal* command.

Option	Displayed information
-y	turns on request tracing for <i>device</i>
-n	turns on request tracing for <i>device</i>
-p	continuously displays traced requests for <i>device</i>
-t	prints a table of traceable devices
-v	is verbose when printing traceable devices using -t
-m <i>num</i>	For efficiency, <i>iotal</i> waits until <i>num</i> requests have occurred before displaying them. The default is 5 requests.

By default, *iotal* will display requests as:

sssssss + ddddddd = tttttt C llllll OOOOOOOO

The meaning is:

sssssss
length of time the request remained queued in the device driver before it was passed on to the device

ddddddd
length of time the device took to complete the request

ttttttt
sum of sssssss and ddddddd

C is R if it was a read or receive request.
is W if it was a write or send request

11111111
length of the request in bytes

00000000
device-specific information that may be present for some types of devices. For disks the format of this information is as follows:

[*cccc:hhhh:ssss*]

cccc
cylinder number

hhhh
head or track number

ssss
sector

3.9 tunestat(1M)

The *tunestat* command prints the resource usage of structures that are allocated based upon tunable parameters in *mtune*(4) and *stune*(4). *tunestat* is actually a shell script that uses *crash*(1M) and *ikdb*(8) to retrieve most of the information. It has the following synopsis:

/usr/sbin/tunestat [-n *namelist*] [-p *percent*] [-q] [*variable ...*]

The options mean:

Option	Displayed information
-n	The text file <i>namelist</i> contains the symbol table information needed for symbolic access to the system memory image to be examined. The default <i>namelist</i> is <i>/stand/unix</i> .
-p	The Option is used to print out only those structures whose peak usage compared to the maximum is greater than or equal to <i>percent</i> .
-q	The option invokes the quiet mode. The table header and errors will not be printed.

Example:

You want to print the statistics for *tune_proc*, the number of processes in the system:

```
# tunestat tune_proc
```

The output would be:

Variable	Current	Peak	Maximum	Peak %	Tunable Parameters
tune_proc	36	40	4116	0	"NPROC"

The meaning is:

Variable
name of the variable that tracks the structure.
The name is in the form of *tune_structure-name*.

Current
aktual value

Peak
high water mark since the kernel was booted

Maximum
maximum allowed by *mtune(4)* or *stune(4)*

Peak %
percentage of the high water mark versus the maximum

Tunable Parameters
associated tunable parameter(s)

If no variables are given, all the structures that the *tunestat* command knows about will be printed. If a variable is not a valid name, an error message will be printed and the command will continue with the rest of the variables.

3.10 truss(1)

You can use the *truss(1)* command to examine system calls and signals. The command has the following syntax:

```
truss [option] ... command
```

truss executes the specified command and logs

- the system calls which are activated by the command,
- the signals which it receives,

- the machine errors it causes.

The example below illustrates *truss* operation:

```
# truss date
```

Example output:

```
execve(../sbin/date", 0x000000FFFFFEFC10, 0x000000FFFFFEFC20)  argc = 1
access(../usr/lib/locale_new/En_US.ASCII/LC_MESSAGES", 0) = 0
access(../usr/lib/locale_new/En_US.ASCII/LC_MESSAGES", 12) = 0
access(../usr/lib/locale_new/En_US.ASCII/LC_MONETARY", 0) = 0
access(../usr/lib/locale_new/En_US.ASCII/LC_MONETARY", 12) = 0
access(../usr/lib/locale_new/En_US.ASCII/LC_COLLATE", 0) = 0
access(../usr/lib/locale_new/En_US.ASCII/LC_COLLATE", 12) = 0
access(../usr/lib/locale_new/En_US.ASCII/LC_TIME", 0) = 0
access(../usr/lib/locale_new/En_US.ASCII/LC_TIME", 12) = 0
access(../usr/lib/locale_new/En_US.ASCII/LC_CTYPE", 0) = 0
open(../usr/lib/locale_new/En_US.ASCII/LC_CTYPE", 0_RDONLY, 0) = 3
read(3, "\0                               h (... , 521)      = 521
close(3)                                           = 0
sysconf(PAGESIZE)                                = 16384
brk(0x00000000004690E0)                           = 0
access(../usr/lib/locale_new/En_US.ASCII/LC_NUMERIC", 0) = 0
open(../usr/lib/locale_new/En_US.ASCII/LC_NUMERIC", 0_RDONLY, 0) = 3
fxstat(2, 3, 0x000000FFFFFED310)                  = 0
read(3, " . \0", 3)                               = 3
close(3)                                           = 0
time()                                             = 885388238
access(../usr/lib/locale_new/En_US.ASCII/LC_TIME", 0) = 0
open(../usr/lib/locale_new/En_US.ASCII/LC_TIME", 0_RDONLY, 0) = 3
fxstat(2, 3, 0x000000FFFFFED860)                  = 0
read(3, " J a n\ n F e b\ n M a r\ n"... , 291)    = 291
close(3)                                           = 0
ioctl(1, TCGETA, 0x000000FFFFFED9F4)              = 0
Wed Jan 21 14:10:47 MET 1998
write(1, " W e d   J a n   2 1   1"... , 29)        = 29
_exit(0)
```

Each line in the log contains the following information:

- a system call with its arguments and return values
- the name of a signal

If the *-c* option is specified, *truss* does not output the logged system calls, errors and signals. Instead of this, it counts the number of occurrences and outputs the result as soon as the specified command has terminated or *truss* is interrupted. This result contains individual totals for system calls, errors and signals. For example:

```
# truss -c cp /unix /tmp
```

Example output:

syscall	seconds	calls	errors
_exit	.00	1	
read	.37	1141	
write	.51	1138	
open	.00	3	
close	.00	4	
creat	.03	1	
brk	.00	1	
access	.00	11	
ioctl	.00	1	
execve	.00	1	
umask	.00	1	
fstatvfs	.00	2	
xstat	.00	3	
fxstat	.00	1	
sysconfig	.00	1	
	----	----	----
sys totals:	.91	2310	0
usr time:	.00		
elapsed:	1.57		

Using this information you can, for example, ascertain the data volume which is transferred per I/O operation. In the above example, the size of the */unix* file was 5534845 bytes. From this it can be deduced that $5534845 / 676 = 8$ Kbytes were transferred per *write* system call.

3.11 proceso(1M)

proceso(1M) reads the processes and kernel segments page tables. These page tables hold the page table entries for each page in the address space of processes and the kernel. By checking the individual flags of the page table entries, *proceso* can determine which pages are resident in the main memory and whether the pages are private to that process or shared with other processes. System processes cannot be analyzed.

The user's access permissions determine how the individual processes can be accessed.



proceso may yield incorrect results or even core dump if the system is not booted from */unix*. So please verify that */unix* is linked to the kernel image that has been booted.

The command has the following syntax:

proceso [*option*]

The table below lists the options which can be specified for the *proceso* command. If no option is specified then the command behaves as if the *-s* switch had been specified. When specifying the *-p* option, *proceso* will also analyze the processes memory usage.

Option	Displayed information
-s	Summary information about memory usage
-p	Summary information about all processes Detailed information you get with <i>-pv</i> or <i>-pvv</i>
-k	Detailed information about each kernel segment
-M	Physical memory usage per process
-c	Running and ready-to-run processes Detailed information you get with <i>-cv</i> or <i>-cvv</i>
-u	Scans the file <i>proceso.ref</i> for memory mapped files and updates the device and inode entries Use this option if file systems were mounted differently or if you use a <i>proceso.ref</i> file from another machine
-uu	Scans all mounted file systems for all entries that have not yet been assigned in <i>proceso.ref</i>  This may take a long time
-r	Scans all directories listed in the <i>proceso.dirs</i> file and adds entries to <i>proceso.ref</i> file

Used files

/opt/bin/proceso/proceso.ref
This file contains all filenames, device and inode numbers of memory mapped files. *proceso* needs this cross-reference list to find out the file names of memory mapped files during runtime.

When *proceso -pv* is executed, an entry with the device and inode number is added to the file *proceso.ref* for files still without cross references. The relevant filename is completed in *proceso.ref* with *proceso -u*.

`/opt/bin/proceso/proceso.dirs`

This file contains the directory names of all binaries and libraries that may be used by the running processes. You may also add directory entries for specific add-on products (databases, etc.). See also the `-r` option.

If the `proceso.ref` file was removed or cleared for some reason, you can create a new reference file based on the directories listed in `proceso.dirs`.

`/opt/bin/proceso/proceso.sect`

The file is only relevant for the `M` option. For selected physical memory areas (e.g. CPU board memory), it contains the following entries: the Id for the area, its hexadecimal start address and its hexadecimal length.

If `0` is entered as the start address for subsequent entries, the start address is the `end address+1` of the previous entry. If the `proceso.sect` file is in the current directory it will be used.

Information on how the memory is assigned to the different boards you can display with `showconf -M`.

Determining the memory occupied by a process

Use first `proceso -r` to create the cross-reference list `proceso.ref`.

Than You can use `ps -e | proceso -pv` to determine the amount of memory a process physically occupies.

Example output:

ReliantUNIX-Y wait 5.45 A1010 RM600 8/8192 R10000 --- Mon Jan 04 10:28:40 1999

***** ANALYSIS OF SYSTEM MEMORY USAGE BY PROCESSES *****

pid	process	private core total	shared core total	files core total	shared memory core total	cmd
1858		0 368	0 1024	0	0	/usr/sbin/in.timed -M
2481		0 368	0 960	0	0	/usr/lib/saf/ttymon -
g -p	Console Login: -d /dev/console -l console					
320		0 512	0 752	0	0	log3analyze -t0
319		0 192	0 656	0	0	/opt/log3/bin/log3svr
-ed0 -m1440						
2480		80 256	96 896	0	0	/usr/lib/saf/sac -t 3
00						
164		816 1648	176 208	0	0	/sbin/errlogd -a
177		864 1040	128 1024	0	0	/sbin/errord
1725		0 160	0 592	0	0	/usr/sbin/silsd
.	.	.	.	.	.	.
.	.	.	.	.	.	.

total 1592112 9165200 KB				
***** LIST OF MEMORY MAPPED FILES *****				
dev(dec)	ino(dec)	core (KB)	total (KB)	name
1835008	8719	80	80	/sbin/date
1835008	8740	0	288	/sbin/killall
1835008	8787	0	32	/sbin/ldcontroller
1835008	8809	16	32	/sbin/rc0
1835008	8827	224	224	/sbin/ups2
1835008	8839	0	144	/sbin/dkconfig
1835010	1902	0	80	/opt/bin/sim/simmond
1835010	24165	0	256	/usr/bin/X11/xdm
.	.	.	.	.
total 1728 10016 KB				
***** SUMMARY OF SYSTEM MEMORY USAGE *****				
type	number	core(KB)	total(KB)	
process private	438	74044	132964	
mapped files	112	18912	24132	
kernel memory		74384	186636	
memory disk	1	65536		
shared memory	0	0	0	
not ref. pages		48		
kernel size		15695		
free memory		56256		
proceso TOTAL		304875		
physical memory		327680		

The columns mean:

pid

process ID



proceso handles the first column of the *ps* output as the process ID. For this reason, only *ps* options that output the process ID in the first column should be used.

core

size of main memory used

total

virtual size of main memory



The final total may differ by up to 10 percent from the physical memory size available in the system.

cmd

process name

name

file name

If “—” is output here instead, this indicates that there is as yet no entry in *proceso.ref* for the relevant files. *proceso.ref* can be completed using the option *-uu*.

The address space may consist of three sections:

process private

This type of memory is used solely by that process. This is typically initialized data, memory allocated using *malloc(3C)*, and the stack segment.

shared files

UNIX supports the concept of memory mapped files. This is a mechanism by which the data of a file is mapped into the address space of a process. This means a process can access this data by using an address operation (see also *mmap(2)*). Examples of such memory mapped files are the code segments of an executable or a library.

shared memory

Shared memory is a mechanism of the interprocess communication facilities (IPC) of UNIX. Shared memory processes can communicate and exchange data very quickly.

3.12 xload

The X11 client *xload* is a component of the SINIX/windows User Environment. This produces a time-related histogram of the CPU queue. This histogram depicts the so-called “system load” as it is output by the *uptime(1)* command. This time-related CPU queue specifies the number of processes which are currently active and waiting in a queue for processing. The greater the CPU load, the longer the queue. The length of the queue is therefore an indication of the system’s CPU load.

There are two ways of starting *xload* under SINIX/windows:

- via *Application Manager – Desktop-Tools – System Load*
- directly from a shell

The command syntax is as follows:

xload [-*geometry* *width* x *height*] [-*update* *seconds*]

The option *-geometry* determines the size of the *xload* window. You may enter the required number of pixels for *width* and *height*. You may also subsequently modify the preset value (120x139) using the resizing border of the *xload* window. The *-update* option specifies the update frequency. Each time the specified period elapses, *xload* draws a line which is 1 pixel wide on the histogram. If this option is not specified then a new line is drawn by default every 10 seconds.

Careful choice of the width of the *xload* window and the update frequency enables you to record the activity of, for example, the last hour or last 24 hours in a single window.

Example:

```
# xload -geometry 360x180 -update 10 &
```

displays the next hour
(you do not need to specify the
-update 10 option.)

```
# xload -geometry 576x180 -update 150 &
```

displays the next 24 hours

Example output:

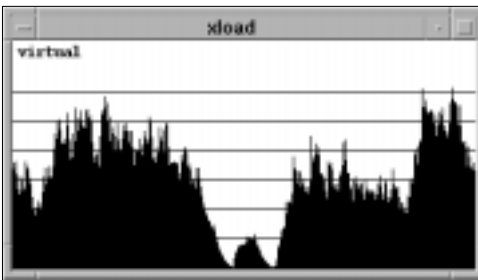


Figure 8: Displaying the system load with *xload*

For further information on *xload* refer to the online manual page under SINIX/windows.

3.13 SINIX/monitor

The Performance Monitor, *SINIX/monitor*, is used for the graphic presentation of important system performance values. If bottlenecks are detected, a visual warning is also issued. Detail windows list the most important sources of bottlenecks together with their level of utilization. *SINIX/monitor* is integrated into the desktop of the SINIX/windows User Environment. It can also be operated via the TRANSVIEW Control Center.

SINIX/windows operation is described in the manual “SINIX/windows User Environment V3.0 – Introduction to Handling and Configuration”. If you are familiar with the operation of Microsoft Windows[®], you will soon be able to work with SINIX/windows and *SINIX/monitor*.

Of course, the *SINIX/monitor* itself contributes to CPU load. However, the resulting distortion of performance values is low: at an interval of 5 seconds, the additional CPU load is less than 5%. If the LAN display is active, this value increases to 10% (measured at an RM400).

Requirements

- Reliant UNIX
- SINIX/windows as of version 2.0
- 2 Mbytes free main memory
- 300 Kbytes free storage space in the */opt* directory.
- Graphics terminal (e. g. an X terminal)

You can only enjoy full *SINIX/monitor* functionality (e.g. colored labeling of bottlenecks) if you use a color X terminal.

Installation

Install the *SImon* package from the CD-ROM named “CD-SYS-MI”. You can do this via the *SYSADM* user interface. For more information, refer to your computer’s Installation Guide (chapter “Processing software with *SYSADM*”). By default, *SINIX/monitor* is installed under */opt/monitor*. Enter the */opt/monitor/bin* directory in the *\$PATH* environment variable.

Three online manual pages are available for *SINIX/monitor*: *simon(1)*, *simond(1)* and *simond(4)*. These manual pages are located in the */man/En* subdirectory of the *SINIX/monitor* home directory. If you wish to use these pages, you must ensure that the complete pathname is entered in the *\$MANPATH* environment variable (see *man(1)*).

Configuration

You can use two files to configure *SINIX/monitor*:

- */etc/default/simond* contains the parameters for the *SINIX/monitor* daemon, in particular the limit values for warnings and alarms. All users can view these limit values via the menus *Options – List Limits...* (see page 71). However, only the system administrator is able to modify these limit values. To obtain more information at shell level, refer to the manual pages *simon*(1) and *simond*(1) and the *SINIX/monitor* help window.
- */usr/lib/X11/app-defaults/Simon* contains the parameters for the graphical presentation of the measurement results. These parameters specify, for example, which measurement diagrams will be displayed and which colors will be used for normal values, warnings and alarms. All users can modify these parameters. To do this, copy the above-mentioned file to your *\$HOME* directory and edit as required. To obtain more information at shell level, refer to the manual page *simon*(1) and the *SINIX/monitor* help window.

All users can override the specified parameters if they do not start *SINIX/monitor* as an icon but from a shell (see below).

Starting SINIX/monitor

- Start *SINIX/monitor* from the SINIX/windows desktop:

Double click successively on the following icons, starting in the *Main Desktop* window:

Applications

The *Main/Applications Desktop* window appears

SINIXmonitor

- Start *SINIX/monitor* from a shell:

Switch to the “normal” shell and enter the *simon* command. The command syntax is as follows:

simon [-base_option ...] [-add_option ...]

You can specify a large number of options to influence *SINIX/monitor* operation. For example, you can specify,

- which measurement diagrams the *SINIX/monitor* main window is to contain,
- which limit values are to apply for warnings and alarms,
- which colors are to be used for warnings and alarms.

This overrides the settings in the files */etc/default/simond* and */usr/lib/X11/app-defaults/Simon*. For a detailed description of all the options refer to the manual page *simon(1)*.

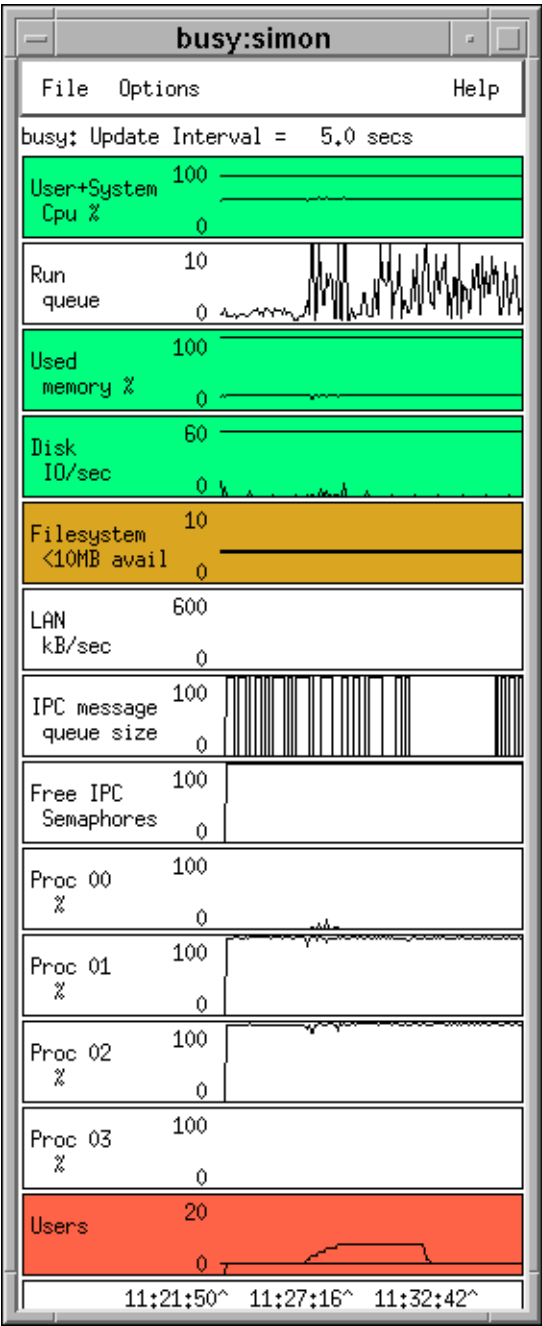
Exiting SINIX/monitor

There are again two ways to exit *SINIX/monitor*:

- Via the window menu header in the title bar of the main menu (see next page):
 - ▶ Click on the menu item *Close*
or press the key combination **Alt F4**.
- Via the menu item *File* in the menu bar of the main window (see next page):
 - ▶ Click on the menu item *Quit*
or press the key combination **Ctrl Q**.

3.13.1 Main window

When you start *SINIX/monitor* the main window appears.



Available detail window
“Leading values for

- “... CPU”
- “... Used Memory”
- “... Disk I/O”
- “... Filesystem available”

Figure 9: Main window of SINIX/monitor

The main window consists of the following sections:

- Title bar (host name of the computer)
- Menu bar (*File*, *Options* and *Help* menu entries)
- Header (update interval)
- Measurement diagrams (the number depends on configuration)
- Footer (time of the measured value to which the “^” is pointing)

When the measurement diagrams reach the right-hand edge of the window, they are moved far enough to the left to permit the display of the current value. When this is done the oldest measured value is deleted.

If you vertically enlarge the *SINIX/monitor* window, the measurement diagrams are also proportionally enlarged. Where necessary, this permits better monitoring of critical values. If you horizontally enlarge the *SINIX/monitor* window you can monitor the values over a longer period.

The measurement diagrams provide a graphic overview of the following performance values:

Measurement diagram	Displayed information	Matches command	
User+System CPU %	CPU load in percent (equals total of %usr + %sys)	sar -u	115
Run queue	Number of processes in the queue (no detail window available)	sar -q	85
Used memory %	Allocated main memory in percent (in contrast, sar displays the free memory)	sar -r	93
Disk I/O/sec	Average number of I/O operations at hard disks (equals total of pread/s + pwrit/s)	sar -b	120
Filesystem < 10MB avail	Free memory in the file systems with fewer than 10 Mbytes free	df -k	114
LAN kB/sec	LAN transfer rate in Kbytes per second (no detail window available; the diagram is only displayed if it has been specially configured)		

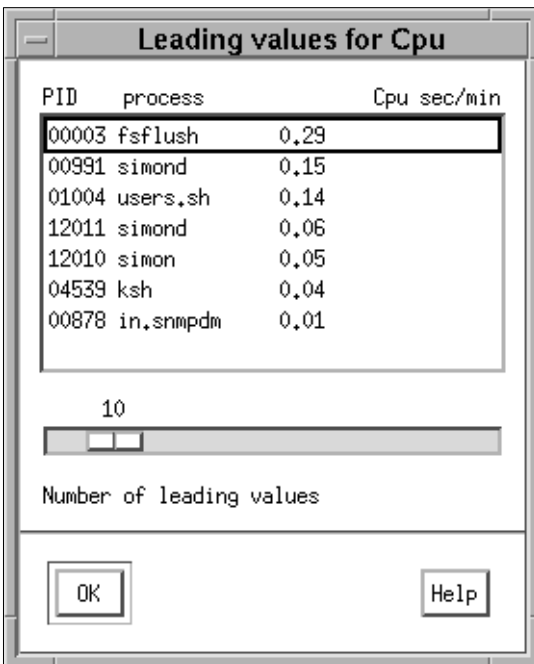
Table 4: Commands corresponding to the SINIX/monitor display

3.13.2 Detail windows

Detail windows are available for some of the measurement diagrams which are displayed in the main window. If you want to call one of these detail windows, simply click the mouse pointer on the corresponding measurement diagram.

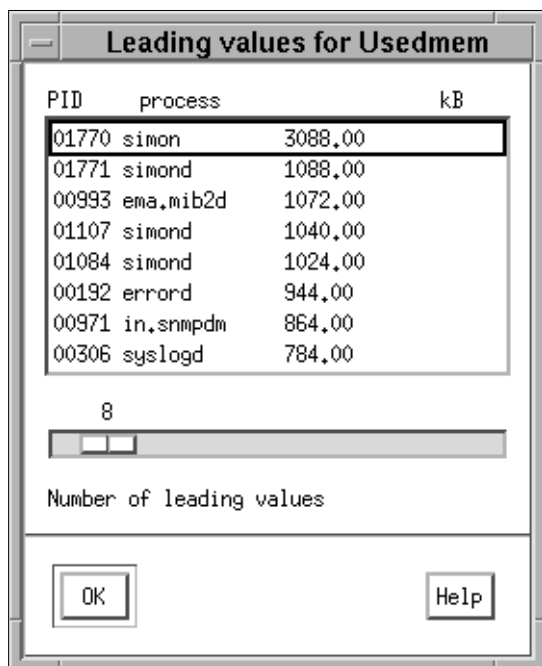
The detail windows are listed in order of importance. The major causes of bottlenecks are listed together with their load statistics. You can use the slide at the bottom of the window to specify the number of items that are to be displayed. The maximum available display corresponds to all the items which have a value other than 0. Open detail windows are updated at one minute intervals.

Detail window “Leading values for Cpu”



CPU utilization during the last minute displayed for each process. Each process is identified by its process (PID) and name.

Detail window “Leading values for Usedmem”



The screenshot shows a window titled "Leading values for Usedmem". Inside, there is a table with three columns: "PID", "process", and "kB". The table lists several processes and their memory usage in kilobytes. Below the table, there is a numeric input field containing the value "8", a horizontal slider bar, and the text "Number of leading values". At the bottom of the window are two buttons: "OK" and "Help".

PID	process	kB
01770	simon	3088,00
01771	simond	1088,00
00993	ema.mib2d	1072,00
01107	simond	1040,00
01084	simond	1024,00
00192	errord	944,00
00971	in.snmpdm	864,00
00306	syslogd	784,00

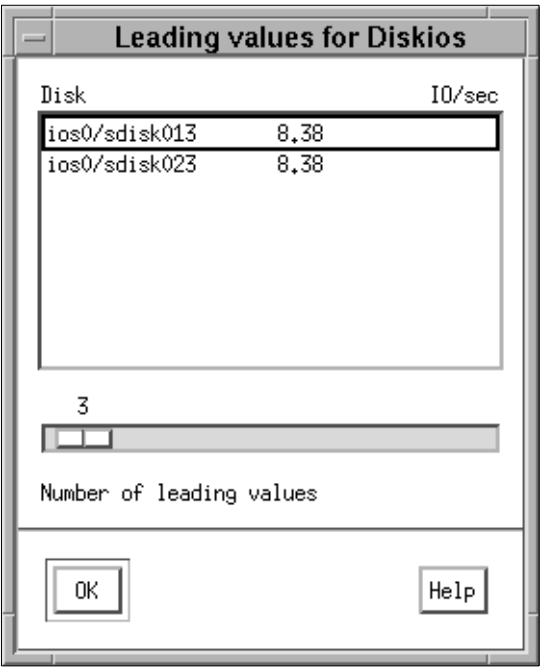
8

Number of leading values

OK Help

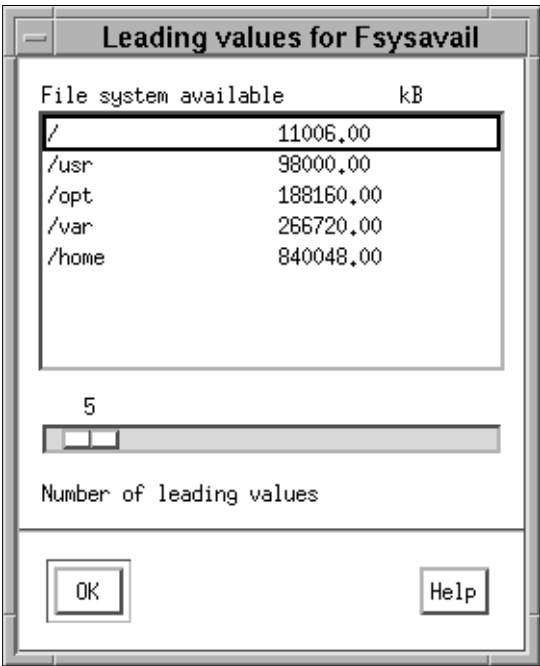
This lists the number of Kbytes of virtual main memory currently occupied by each process. All processes are identified by PID and name.

Detail window “Leading values for Diskios”



This window lists the disks most affected by read and write accesses. It displays the number of accesses per second. Each disk designation corresponds to an entry in the `/dev` directory.

Detail window “Leading values for Fsysavail”



This window lists the most important file systems and specifies how many Kbytes are still free in each file system. The 10% which are still present when the *Filesystem full* messages appears is already deducted from the values displayed here.

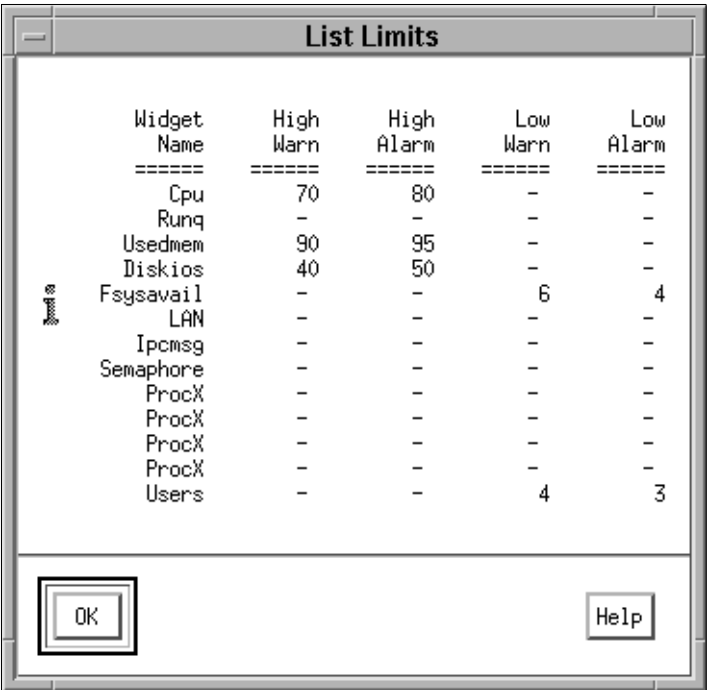
Some file systems may be irrelevant for measurement purposes. There is, for example, little point in always displaying the high level of occupation of a read-only file system. You can therefore exclude such file systems from measurement. To do this select the *File Systems...* menu item from the *Options* menu (see page 73).

3.13.3 Options window

The *Options* item in the menu bar of the main window allows you to access the following options:

- List Limits... (alternative key combination: **Ctrl** **L**)
- Interval... (alternative key combination: **Ctrl** **I**)
- File Systems... (alternative key combination: **Ctrl** **F**)
- Configuration... (alternative key combination: **Ctrl** **C**)

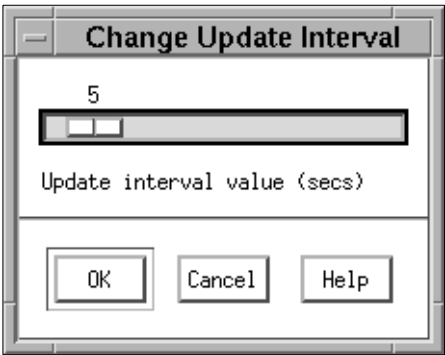
Options window: “List Limits”



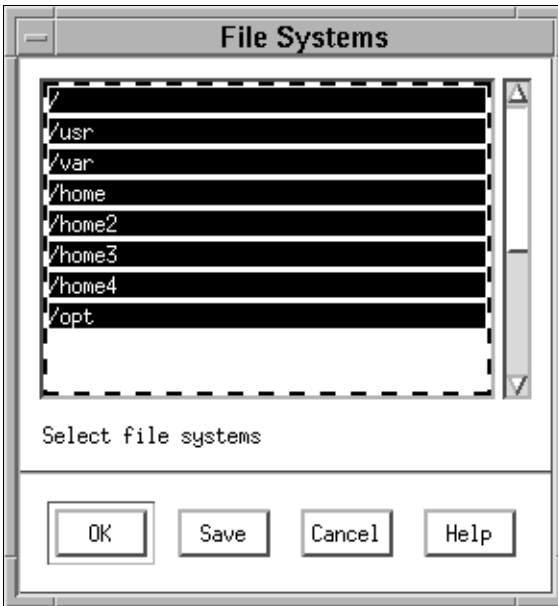
This window displays the limit values for warnings and alarms. “-” means that no limit value has been configured.

If the detected value is higher than the maximum value or lower than the minimum value then the corresponding value in the *SINIX/monitor* main window is displayed in a different color. For a description of how to configure limit values and colors, see page 61.

Options window: “Change Update Interval”



In this window you can use a slide symbol to set the measurement interval to a value between 1 and 60 seconds (presetting: 5 seconds). The system load resulting from measurement operations changes accordingly.

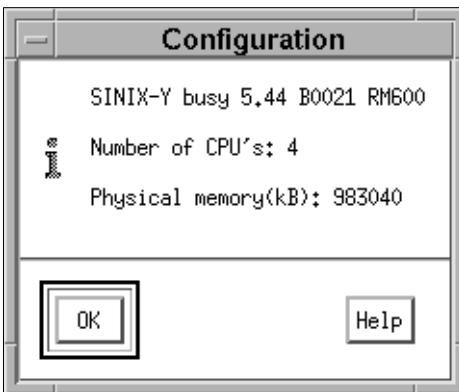
Options window: “File Systems”

You use this window to specify the file systems whose fill level is to be displayed in the *Filesystem <10MB* avail measurement diagram. The lines displayed in reverse video are selected.

Terminate this window as follows:

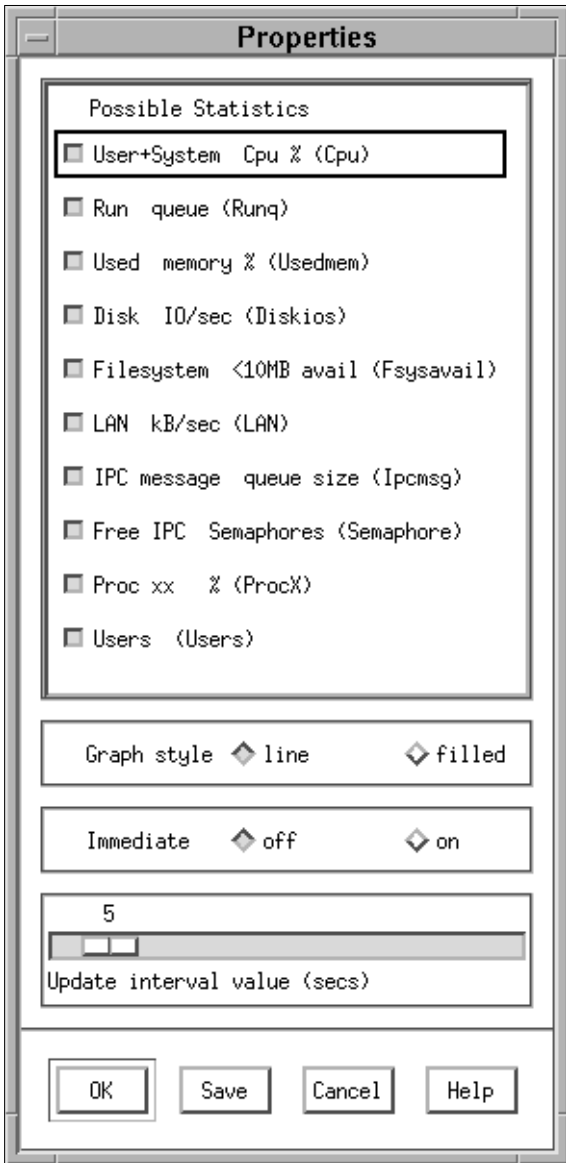
- with **OK** if your selection is to apply only to this session,
- with **Save** if your selection is to apply to future sessions as well.

If you exit the window with **Save**, your selection will be stored in the *\$HOME* directory of the user who called the function.

Options window: “Configuration”

This window displays the same information as is provided by the *uname -a* command (see section “Operating system version” on page 80).

Options window: “Properties”



Here you can specify

- which measurement diagrams should be displayed in the *SINIX/monitor* main window
- how often these diagrams should be updated

Terminate this window as follows:

- with **OK** if your selection is to apply only to this session,
- with **Save** if your selection is to apply to future sessions as well.

If you exit the window with **Save**, your selection will be stored in the *\$HOME* directory of the user who called the function.

3.13.4 Help window

When using *SINIX/monitor* you are supported by comprehensive English help texts. There are two ways of accessing this help:

- Click the mouse pointer on *Help* in the menu bar of the main window. This opens a pull-down menu in which you can select the required item.
- Click the mouse pointer on the **[Help]** button. This button is available in the detail windows and in the options windows.

You can manipulate the Help window like any other window. Click the mouse pointer on an emphasized keyword in order to obtain further information.

4 Shell procedures for load measurement

If you want to conduct an overall system analysis, you must perform as complete a measurement as possible of all system activities during a period of high system load. You must then analyze the data which you have measured. You should document the system load, environment and measuring procedure precisely. Only in this way can you test whether system performance is improved following a modification.

In particular, you should measure and log the following information:

Information	Commands
Process activity	<i>ps(1)</i>
IPC activity	<i>ipcs(1)</i>
Memory activity	<i>sar(1)</i> , <i>vmstat(1M)</i>
Disk activity	<i>sar(1)</i> , <i>dkstat(1M)</i>
CPU load	<i>sar(1)</i> , <i>mpstat(1M)</i>
Paging/swapping activity	<i>sar(1)</i>
System call statistics	<i>scstat(8)</i>

Table 5: Commands for load measurement

4.1 Overview of shell procedures

This chapter presents a range of shell procedures which you can use to perform or analyze the above-mentioned measurements. Please note that these procedures are only intended as examples. You will need to adapt these procedures to your actual applications if you are to obtain meaningful measurement results. You may also use other measurement methods if necessary.



This Tuning Guide is also available online via the Internet. You can copy the shell procedures from there. The address is:

<http://www.sni.de/cgi-bin/nph-dynaweb/docs/collections.htm>

Shell procedures for load measurement		Command used	Files containing measurement results	Shell procedures / commands for evaluation	
Name				Name	
<i>ps_meas</i>	71	<i>ps -avx</i>	<i>ps.dat</i>	<i>ps_stat_eval</i>	86
				<i>ps_time_eval</i>	87
				<i>ps_size_eval</i>	105
		<i>ipcs -qa</i>	<i>ipcs_msg.dat</i>	<i>cat, vi</i>	127
		<i>ipcs -ma</i>	<i>ipcs_shm.dat</i>	<i>cat, vi</i>	130
		<i>ipcs -sa</i>	<i>ipcs_sem.dat</i>	<i>cat, vi</i>	129
<i>vm_meas</i>	72	<i>vmstat</i>	<i>vm.dat</i>	<i>vm_mem_eval</i>	92
<i>dk_meas</i>	72	<i>dkstat</i>	<i>dk.dat</i>	<i>dk_disk_eval</i>	117
<i>main_meas</i>	74	<i>sar</i>	<i>sar_bin.dat</i>	<i>sar</i>	div.
			<i>sar_asc.dat</i>	<i>cat, vi</i>	
		<i>dkstat -p</i>	<i>dk-p.dat</i>	<i>cat, vi</i>	39
		<i>dkstat -g</i>	<i>dk-g.dat</i>	<i>cat, vi</i>	39
		<i>dkstat -b</i>	<i>dk-b.dat</i>	<i>cat, vi</i>	38

Table 6: Overview of proposed shell procedures

The procedure *main_meas* starts the procedures *ps_meas*, *vm_meas* and *dk_meas*. The files containing the measurement results are generally saved to the directory *\$HOME/measure*.

4.2 Shell procedure `ps_meas`

During load measurement it is advisable to log all processes periodically in a file. This should be performed independently of the period and objective of the measurement. In this way you can then identify not only the point at which highest load level occurred but also the processes which caused it. Equally, you should periodically log all IPC activities (IPC = InterProcess Communication: message queues, semaphores, shared memory) in a separate file.

To do this you can use the `/usr/ucb/ps -avx` command. Avoid selecting too short an interval since *ps* itself is very processor intensive.

 At high system loads and with a large number of active processes, the *ps* call should not run periodically in the background since it is extremely likely that the call will not succeed. Generally speaking, a *ps* call is “expensive” (in terms of CPU load). For this reason it should not be issued too frequently, for example not every 5 seconds. In high-load systems, even an interval of 30 seconds may be too short.

The example *ps_meas* procedure logs the following at 30 seconds intervals:

- Process activity in the file `$HOME/measure/ps.dat`
- Message queue evaluations in the file `$HOME/measure/ipcs_msg.dat`
- Semaphore evaluations in the file `$HOME/measure/ipcs_sem.dat`
- Shared memory evaluations in the file `$HOME/measure/ipcs_shm.dat`

```
# Procedure: ps_meas

if [ ! -d $HOME/measure ]
then mkdir $HOME/measure
fi

PSFILE=$HOME/measure/ps.dat
MSGFILE=$HOME/measure/ipcs_msg.dat
SHMFILE=$HOME/measure/ipcs_shm.dat
SEMFILFILE=$HOME/measure/ipcs_sem.dat

>$PSFILE
>$MSGFILE
>$SHMFILE
>$SEMFILFILE

while :
do
    date >> $PSFILE
    /usr/ucb/ps -avx >>$PSFILE    # Omit if there are many processes!
                                # otherwise value of ps data reduced.
    ipcs -qa >>$MSGFILE
    ipcs -ma >>$SHMFILE
    ipcs -sa >>$SEMFILFILE
    sleep 30
done
```

Start the procedure in the background:

```
# ps_meas &
```

The measured data obtained by the procedure are written to different files. No output to *stdout* is generated during processing. When measurement is complete the procedure must be ended with *kill*(1).

4.3 Shell procedure `vm_meas`

You can use the `vmstat(1M)` command to generate memory statistics. The example of the `vm_meas` procedure below writes the output of the `vmstat` command to the file `$HOME/measure/vm.dat` for subsequent analysis. No output to `stdout` is generated.

```
# Procedure: vm_meas

if [ ! -d $HOME/measure ]
then mkdir $HOME/measure
fi

INTERVAL=1
NUMBER=2
VMFILE=$HOME/measure/vm.dat

echo $INTERVAL > $VMFILE
date >> $VMFILE
/usr/ucb/vmstat $INTERVAL $NUMBER >> $VMFILE
```

Start the procedure in the background:

```
# vm_meas 10 120 &
```

For an example of the contents of the file `$HOME/measure/vm.dat` see page 43. The analysis of the measured results and the output from the procedure `vm_meas` is described in the section “Paging activity” on page 98.

4.4 Shell procedure `dk_meas`

The example `dk_meas` procedure writes the output from the `dkstat(1M)` command to the file `$HOME/measure/dk.dat` for subsequent analysis. No output to `stdout` is generated.

```
# Procedure: dk_meas

if [ ! -d $HOME/measure ]
then mkdir $HOME/measure
fi

INTERVAL=1
DKFILE=$HOME/measure/dk.dat

echo $INTERVAL > $DKFILE
date >> $DKFILE
dkstat -z                                # delete old dkstat statistics
exec dkstat $INTERVAL >> $DKFILE
```

The `dkstat` command is unable to terminate itself by passing a parameter (counter). Consequently, when measurement is complete the procedure must be ended with `kill`. Calling the procedure via `exec` prevents the generation of a child process and facilitates termination from the measuring procedure itself.

Start the process in the background:

```
# dk_meas 10 &
```

Example contents of the `$HOME/measure/dk.dat` file:

```
10
Wed Jan 13 11:00:55 GMT 1999
Average Disk Transfers (sampled every 10 seconds)
 0s000  0s001  0s010  0s002  0s011  0s012
    0      0      0      0      0      0
    0      0      0      1      0      0
    0      0      2     12      2      1
    3      0      3     30      3      0
    4      0      1     41      1      0
    3      0      7     17      2      0
    2      0      2      2      3      0
    0      0      4      6      5      0
    1      0      3      2      3      0
    0      0      0      0      0      0
    2      0      0      0      0      0
    0      0      2      0      4      0
    0      0      5      3      2      0
    2      0      4     12      2      1
    1      0      5      7      3      0
```

The analysis of the measured results and the output from the `dk_meas` procedure are described in the section “Determining the disk load” on page 114.

4.5 Shell procedure `main_meas`

In the example below, the application is measured for a period of 20 minutes. Process activity is logged every 30 seconds and system activity measured every 10 seconds. This latter value results in a total of 120 measurements. The measured values are written to the binary file `$HOME/measure/sar_bin.dat` for subsequent analysis.

Measurement is performed by the shell procedure `main_meas` within the above framework:

- Process activity is logged.
- Memory statistics are logged.
- Input/output statistics for the disks are logged.
- System activity is measured.

- certain *dkstat* statistics are measured.
- When measurement is complete, the logging procedures are terminated.
- The binary file containing all *sar* output is converted into an ASCII file.

```
# Procedure: main_meas

INTERVAL=10                      # Time in seconds
NUMBER=120                       # Number of intervals

LANG=en_US.ASCII
export LANG
if [ ! -d $HOME/measure ]
then mkdir $HOME/measure
fi

echo "START of measurement"

sh ps_meas &                    # Log process activity
PSPID=$!

vm_meas $INTERVAL $NUMBER &    # Memory statistics
dk_meas $INTERVAL &            # Disk statistics I/O's over time
DKPID=$!

sar -o $HOME/measure/sar_bin.dat $INTERVAL $NUMBER
                                # Create sar-statistics
kill -9 $PSPID $DKPID          # End procedures
sar -A -f $HOME/measure/sar_bin.dat > $HOME/measure/sar_asc.dat
                                # convert to ASCII file
dkstat -p > $HOME/measure/dk-p.dat # Disk statistics at partition level
dkstat -g > $HOME/measure/dk-g.dat # Disk statistics read/writes, bytes
dkstat -b > $HOME/measure/dk-b.dat # Disk statistics block sizes

echo "END of measurement"
```

INTERVAL=10 defines the interval between measurements in seconds,
NUMBER=120 defines the number of intervals.

The *sar* command also displays the values measured for CPU load on screen. In this way you can at least analyze the CPU load while measurement is progressing. However, you can also disable this output by redirecting the default output to */dev/null*.



The measurement files may be relatively large (in our example: *ps.dat* approximately 180 Kbytes, *sar_bin.dat* approximately 1.5 Mbytes). Before starting measurement, you should therefore check that there is sufficient space available in the selected file system (in our example: */home*). You can do this using the command *df -k* (see page 112).

Start the measurement before starting the application programs:

```
# main_meas                                in foreground with output
# main_meas > /dev/null &                 in background without output
```

Example output for a measurement which is running in the foreground (corresponds to the output from *sar -u*, see page 113):

START of measurement						
SINIX-Y rte 5.45 A1010 RM600 01/04/99						
12:00:55	%usr	%sys	%wio	%idle	int/s	intdef/s
12:01:05	2	4	0	94	6	0
12:01:15	10	15	2	73	24	0
12:01:25	4	8	18	69	31	0
12:01:35	10	15	35	39	67	0
12:01:45	19	32	43	6	93	0
12:01:55	33	40	13	13	63	0
12:02:05	31	16	7	45	32	0
12:02:15	34	21	13	32	180	0
Average	18	12	4	66	93	0
END of measurement						

As soon as measurement has commenced, start the application. If the overall application is slow, you should first measure the individual components. Then start the full application in order to check whether an overload occurs.

As soon as measurement has terminated you can evaluate and analyze the measurement files.

5 Evaluating load measurements

Once you have completed load measurement, you can analyze the results step by step and look for possible system bottlenecks.

5.1 Hardware configuration

You can use the *autoconf -l* command to display your computer's current hardware configuration, e. g.:

```
System Configuration:
sp_bus00
  Cpu-Board0 in slot 0 ABIC-rev 8
    Logical CPU id 00 — Physical id 00 agentno 0 PRId-Impl 9 PRId-Rev 3.4
    Logical CPU id 01 — Physical id 01 agentno 3 PRId-Impl 9 PRId-Rev 3.4
    Onboard Memory: base 0x0 size 448 MB (7x64 MB)
  Cpu-Board1 in slot 2 ABIC-rev 8
    Logical CPU id 02 — Physical id 02 agentno 0 PRId-Impl 9 PRId-Rev 3.4
    Logical CPU id 03 — Physical id 03 agentno 1 PRId-Impl 9 PRId-Rev 3.4
    Logical CPU id 04 — Physical id 04 agentno 3 PRId-Impl 9 PRId-Rev 3.4
    Onboard Memory: base 0x1c0 size 448 MB (7x64 MB)
  EHIO5 cab:3 slot:4 agent:5 — Board-rev 8 ABIC-rev 2
    BLink 00 local
      X86 Bridge node_id:0xf006 — Chip Rev 1
        ios0/scon17 53C770 chan:1
          ios0/sport17 Id: 8 DF 16 Bit 10MHz shared
          ios0/sdisk170 HD14 — HD14
          ios0/sdisk171 HD14 — HD14
          ios0/sgen17b Multi Hosted HA Target
        SRAM 960K Bytes
  EHIO5 cab:3 slot:10 agent:6 — Board-rev 7 ABIC-rev 2
    BLink 00 local
      PCI Bridge (lower cage) node_id:0xf082 — Chip Rev D
        PCI_Bus 00
          zx06 ETHER PENE_V03 DEC21140 V34 [0:c0:95:fa:73:33] Phy 840.1
          SRAM 512K Bytes
          PCI/PCI Bridge onboard
        PCI Bridge (upper cage) node_id:0xf080 — Chip Rev D
          PCI_Bus 00
            zx04 ETHER PENE_V03 DEC21140 V34 [0:c0:95:fa:73:9] Phy 840.1
            fc/fc1 FC LP6502-OS [FW 2.12 00], Top.: FC_AL, state: READY
            iofo/sport.16 FCP
            SRAM 512K Bytes
            PCI/PCI Bridge onboard
          X86 Bridge node_id:0xf086 — Chip Rev 1
            ios0/scon00 53C770 chan:3
              ios0/sport00 Id: 7 SE 8 Bit 10MHz
              ios0/stape002 MC44
              ios0/stape003 MC75
              ios0/sdisk000 PLEXTOR CD-ROM PX-32CS — GENERIC CD-ROM
              ios0/sfdisk001 FD11
```

```

ios0/scon01 53C770 chan:2
  ios0/sport01 Id: 8 SE 16 Bit 10MHz
    ios0/sdisk011 HD02 -- HD02
    ios0/sdisk012 HD14 -- HD14: 5.45 A1010 piece 0
    ios0/sdisk013 HD02 -- HD02
    ios0/sdisk014 HD02 -- HD02
    ios0/sdisk015 HD02 -- HD02
    ios0/sdisk016 HD02 -- HD02
  ios0/scon02 53C770 chan:1
    ios0/sport02 Id: 8 SE 16 Bit 10MHz
      ios0/sdisk021 HD02 -- HD02
      ios0/sdisk022 HD14 -- HD14: 5.45 A1010 piece 1
      ios0/sdisk023 HD02 -- HD02
      ios0/sdisk024 HD02 -- HD02
      ios0/sdisk026 HD02 -- HD02
    zx05 ETHER EIFC-HC.V4 DEC21140 V34 [8:0:6:d:9d:d1] Phy 840.1
    SRAM 960K Bytes
  md/mdisk0 memory disk -- no memory allocated

```

The specifications in the individual lines have the following meaning:

sp_bus00

Central system bus (SPbus = Synchronous Pipelined Bus).

Cpu-Board# ...

Logical ID of a CPU board on the SPbus (#), slot (*slot*) and hardware revision level (*ABIC-rev*)

Logical CPU id ## -- Physical id ## ...

Logical and physical ID of a CPU on the CPU board (##), processor slot on the board (*agentno*) and chip version of the processor (*PRId*).

Onboard Memory ...

Physical base address (*base*) and size (*size*) of the memory on a CPU board

EHIOS ...

Slot (*cab/slot*), logical ID on the SPbus (*agent*) and hardware revision level (*Board-rev/ABIC-rev*) of a board (here: EHIOS)

BLink 00 local

Bus used as the connection between the SPbus and the X86 or PCI bridge

X86 Bridge ...

ID (*node_id*) and hardware revision level (*Chip Rev*) of an X86 bridge between a SCSI controller and the BLink bus

PCI Bridge ...

Location, ID (*node_id*) and hardware revision level (*Chip Rev*) of a PCI bridge between a PCI bus and the BLink bus

`ios0/scon##` ...
Name, type and board slot (*chan*) of a SCSI controller

`ios0/sport##` ...
Name, host adapter SCSI ID (*Id*), connection type (e.g. *DF* = Differential ended, *SE* = Single ended), bus width (*Bit*) and frequency (*MHz*) of a SCSI controller

shared means that the devices on this controller can also be used by other controllers

`ios0/sdisk###` ...
Name and identifier (e.g. *HD14*) of a magnetic disk drive (hard disks and optical disks)

`ios0/sgen###` ...
Name of any SCSI device not supported by Siemens

`ios0/stape###` ...
Name and identifier (e.g. *MC44*) of a magnetic disk drive

`ios0/sfdisk###` ...
Name and identifier (e.g. *FD11*) of a floppy disk drive

`zx##` ...
Name, type, Ethernet address and physical layout of an Ethernet[®] controller

`fc/fc#` ...
Name, type, firmware revision, connection type and status of a Fibre Channel controller

`iof0/sport.##` ...
Name of a Fibre Channel port on the controller `fc/fc#`

SRAM ...
Area on an EHIOS board for storing administration data

`md/mdisk0` ...
Size of the configured memory disk
There is no memory disk configured in the above example.

Use the `autoconf -a` command to obtain additional information about the hardware.

5.2 Operating system version

The `uname -a` command provides the following information about the operating system:

- Name of the operating system
- Node name of the computer
- Version of the operating system
- Revision version of the operating system
- Type of computer
- Number of CPUs/size of the available main memory in Mbytes
- Type of processor

Example output:

```
ReliantUNIX-Y wait 5.45 A1010 RM600 8/8192 R10000
```

Of importance are the operating system version and revision version, here 5.44 A1010. You should always use the current revision version of the software.

The size of the available main memory is calculated as follows:

- Size of the physical main memory (8192 Mbytes)
- Minus size of the memory disk (here 0 Mbytes)



With earlier operating system versions, the `uname(1)` command may return the name “SINIX-Y” or “SINIX-N”. The same applies if you have configured the current revision level of the operating system by means of an update installation. However, you can change the output from the command by setting the environment variable `_SNI_NAME=std`.

5.3 Information about processors

You can use the `uname -a` command to determine the number of processors which are installed in your computer (see above). In multiprocessor computers, you can also use the `autoconf(8)` command for this purpose:

```
# autoconf -l | grep CP | wc -l
```

The messages which are displayed at system start inform you about the processors which are installed.

The file `/var/adm/log/messages` lists the number of processors together with their logical and physical names or numbers in the line starting `Cpu` or `processor`. The messages displayed each time the system boots are appended to this file on each boot operation. This information is also saved in the file `/etc/osm` (also in the case of single processor computers). If you wish you may view these files, for example by using the `cat(1)` command.

```
Processor Configuration:
  Boot processor:      Cpu #0 (p1)
  Other processor(s):  #1 (p2), #2 (p3)
                      #3 (p4), #4 (p5)

CPU 0:  MIPS R10000 Processor Chip Revision: 2.5
FPU 0:  MIPS R10000 Floating Point Unit Revision: 0.0
PClk = 200000000 Hz  SysC1k = 66666666 Hz  SCC1k = 133333333 Hz
Primary Cache Sizes = 32 KB(Instruction), 32 KB(Data)
Secondary Cache Size = 2048 KB
```

You can use the `mpcctl -I` command to obtain more information about the processors which are installed in your computer.

Example output:

CPU	PCPU	STATUS	ATTRS	CPUTYPE	FP1	FP2	CACHSZ	#BND	#XBND	#DRVBND	#PINTBND	CLOCK
0	0	iabxco	iaumd	R100	FMP	-	4096K	0	0	0	0	200
1	1	iabxco	iaumd	R100	FMP	-	4096K	0	0	0	0	200
2	2	iabxco	iaumd	R100	FMP	-	4096K	0	0	0	0	200
3	3	iabxco	iaumd	R100	FMP	-	4096K	0	0	0	0	200
4	4	iabxco	iaumd	R100	FMP	-	4096K	0	0	0	0	200
5	5	iabxco	iaumd	R100	FMP	-	4096K	0	0	0	0	200
6	6	iabxco	iaumd	R100	FMP	-	4096K	0	0	0	0	200
7	7	iabxco	iaumd	R100	FMP	-	4096K	0	0	0	0	200

The columns have the following meanings:

CPU

logical number of CPU

PCPU

physical number of CPU

STATUS

CPU status flags

i(I)

The CPU is (in)active.

a(A)

The CPU is (not) available for normal processing.

b(B)

The CPU is (not) available for normal process conditions.

x(X)

The CPU is (not) available for exclusive process conditions.

c(C)

The CPU cache memory is (in)active.

(The flag is not used in the current Reliant UNIX version. c is always output.)

o(O)

The CPU accepts (no) I/O interrupts.

(The flag is not used in the current Reliant UNIX version. 0 is always output.)

ATTRS

CPU attributes

i(I)

The CPU can accept (no) I/O interrupts.

a(A)

The CPU possesses (no) uniform access to I/O addresses.

u(U)

The CPU possesses (no) uniform access to memory.

m(M)

The CPU is not (is) the master/load CPU.

d(D)

Drivers are (are not) bound to the CPU.

CPUTYPE

CPU type

FP1, FP2

FPU type(s) (FPU = Floating Point Processor Unit)

CACHSZ

Size of secondary cache memory

#BND

Number of processes which are bound to the CPU

#XBND

Number of processes which are exclusively bound to the CPU

#DRVBND

Number of drivers which are bound to the CPU

#PINTBND

Number of interrupts which are bound to the CPU

CLOCK

Internal CPU clock in MHz
The external clock is much less.

5.3.1 Evaluating CPU load

You use the *sar* option *-P* to obtain information about the processor-specific CPU load. You may also evaluate the file *\$HOME/measure/sar_bin.dat* which is generated by the procedure *main_meas* (see page 74):

```
# sar -P -f $HOME/measure/sar_bin.dat
```

Example output:

ReliantUNIX-Y wait 5.45 A1010 RM600 01/04/99										
12:00:55		%usr	%sys	%wio	%idle	wsplck	int/s	intdef/s	pswch/s	%paff
12:01:05	cpu 0 :	0	4	0	96	0	4	0	9	0
	cpu 1 :	4	4	0	92	0	2	0	2	0
	cpu 2 :	14	22	1	63	0	13	0	15	0
	cpu 3 :	6	8	2	84	0	11	0	9	0
	cpu 4 :	5	6	15	74	0	27	0	12	0
	cpu 5 :	3	11	21	65	0	4	0	13	0
Average	cpu 0 :	0	3	0	99	0	7	0	11	0
Average	cpu 1 :	5	2	5	65	0	4	0	32	0
Average	cpu 2 :	18	26	4	61	0	14	0	12	0
Average	cpu 3 :	8	7	5	113	0	55	0	2	0
Average	cpu 4 :	22	8	15	85	0	38	0	19	0
Average	cpu 5 :	4	15	17	57	0	3	0	16	0

The %usr, %sys, %wio and %idle columns must be examined together.

%usr

Proportion of CPU time in user mode

%sys

Proportion of CPU time in system mode

Very high values (~ 100%) mean that the user processes are performing large numbers of, or very intensive, system calls.

%wio

Idle; a process is waiting for block input or output.

%idle

Idle; the CPU is waiting for processes to become executable.

wsplck

Wait time caused by operating system locks (spin locks)

In the current Reliant UNIX version this value is always 0.

`int/s`

Number of interrupts per second

`intdef/s`

Number of delayed interrupts per second

In the current Reliant UNIX version this value is always 0.

`pswch/s`

Number of process switches per second

`%paff`

This parameter is not supported in the current Reliant UNIX version.



If the `%wio` proportion is low (e.g. < 10%) and the `%idle` time is simultaneously equal to 0 for an extended period, then the processors may be overloaded. In this case, precise information can be obtained from the process queue statistics.

The `%usr` and `%sys` values should be approximately equal in a typical multi-user environment. A very high `%sys` value (close to 100%) may indicate the inefficient use of system calls. A typical example of this is the character-by-character accessing of large files.

However, these values may vary greatly between individual applications without this being of great significance. In the case of LAN Manager, for example, it is not unusual for the `%sys` value to reach 90% and the `%usr` value 10% since, in this application, processing of the network protocols is primarily performed by the kernel. However, the reverse is true of processor-intensive applications or databases. In such cases the `%usr` proportion may reach a high level (> 50%).

5.3.2 Evaluating process queue statistics

The evaluation of the process queue statistics can provide additional information about possible processor overload. The `sar` option `-q` provides data about the average length of the queues and the percentage of the time during which the queue was not empty. You may also evaluate the file `$HOME/measure/sar_bin.dat` which is generated by the procedure `main_meas` (see page 74).

```
# sar -q -f $HOME/measure/sar_bin.dat
```

Example output:

ReliantUNIX-Y wait 5.45 A1010 RM600 01/04/99				
12:00:55	runq-sz	%runocc	swpq-sz	%swpocc
12:01:05				%paff
12:01:15	1.4	70		0
12:01:25	1.4	50		0
12:01:35	2.2	50		0
	.	.		.
	.	.		.
	.	.		.
12:20:35	1.3	30		0
12:20:45	1.5	20		0
12:20:55	2.0	10		0
Average	2.0	58		0

Only the columns `runq-sz` and `%runocc` are considered.

`runq-sz`
Average number of executable processes located in main memory and waiting for free CPU time when the run queue is not empty.
The minimum size is always 1, since the *sar* command counts its own process.

`%runocc`
CPU queue load, i.e. the proportion of the time during which at least one process was executable.
The higher this value, the greater the load to which the processors are subjected.

5.3.3 Determining system status of individual processes

You can determine the system status of individual processes by listing all the processes. To do this, use the `/usr/ucb/ps -avx` command. You can then use the *awk* or *egrep* commands to analyse the output from this command.

This also makes use of the example procedure, *ps_stat_eval* below which analyses the file `$HOME/measure/ps.dat`. This file is generated by the procedure *ps_meas* (see page 71) or *main_meas* (see page 74). All processes having the status R, O or X are output together with process number, process name, status and time.

Procedure `ps_stat_eval`

```
# Procedure: ps_stat_eval

awk ' BEGIN {   printf("Process number      ")

                printf("Process name          ")
                printf("Status      ")
                printf("Time\n\n") }
    $5=="GMT" || $5=="MET" || $5=="MST" { alltime = $4
    printf("\n") }
    $3=="R" || $3=="O" || $3=="X" { printf("%-20s", $1)
    printf("%-30s", $7)
    printf("%-10s", $3)
    printf("%s\n", zeit)
    } ' $HOME/measure/ps.dat
```

 The procedure only works if the variable `$5` in line 15 queries the current time zone. The time zones GMT (Greenwich Mean Time), MET (Middle European Time) and MST (Middle European Summer Time) are recognized. If necessary, you may have to modify the corresponding line accordingly.

Start the procedure as follows:

`# ps_stat_eval`

Example output:

Process number	Process name	Status	Time
15359	pf=/usr/sap/C11/SYS/profi	R	12:15:25
15580	/usr/ucb/ps	O	12:15:25
15358	pf=/usr/sap/C11/SYS/profi	R	12:15:25
15359	pf=/usr/sap/C11/SYS/profi	R	12:15:55
15369	P:4096,13,16,	R	12:15:55
15587	/usr/ucb/ps	O	12:15:55
15358	pf=/usr/sap/C11/SYS/profi	R	12:15:55
15369	P:4096,13,16,	R	12:16:25
15594	/usr/ucb/ps	O	12:16:25
15358	pf=/usr/sap/C11/SYS/profi	R	12:16:25

In the same way as when analyzing the process queue statistics, you can once again ascertain which processes are currently waiting for the CPU (status R). The 3rd column in the `/usr/ucb/ps -avx` output specifies the process status. The following specifications are possible:

- O The process is currently being executed.
- S The process is sleeping.

- R The process is runnable.
- I The process is idle.
- Z The process is terminated, but its exit status has not yet been queried with a *wait()* system call (zombie).
- T The process was stopped by means of a signal from the parent process which is monitoring it (traced).

5.3.4 Identifying CPU-intensive processes

If the measuring procedure has periodically started the *ps(1)* command (as in the example procedure *ps_meas* on page 71), you can now evaluate the log file. The 5th column (TIME) of the file *\$HOME/measure/ps.dat* specifies the total execution time in minutes and seconds.

For example you can use the shell procedure *ps_time_eval* to list all the processes which claimed more than one minute of CPU time (TIME).

Procedure *ps_time_eval*

```
# Procedure: ps_time_eval

awk ' BEGIN {   printf("PID          ")
                printf("Process name          ")
                printf("SIZE          ")
                printf("RSS           ")
                printf("TIME          ")
                printf("STAT          ")
                printf("Time\n\n\n") }
  $5=="GMT" || $5=="MET" || $5=="MST" { alltime=$4
                printf("\n\n") }
  $2=="Z" || $5=="SIZE" || $5=="GMT" || $5=="MET" || $5=="MST" { next }
  { glen=length($4)
    tlen=index($4,":")
    tlen == 2 { time=substr($4,1,4)
               min=substr(time,1,1)
               sek=substr(time,3,2) }
    tlen == 3 { time=substr($4,1,5)
               min=substr(time,1,2)
               sek=substr(time,4,2) }
    { size=substr($4, tlen+3, glen)
      rss=$5
      pname=$6
      size="" { size=$5
               rss=$6
               pname=$7 }
      { status=$3
        pid=$1 }
      min>=1 { printf("%-7s", pid )
```

```
printf("%-30s", pname)
printf("%-8s", size)
printf("%-8s",rss)
printf("%-8s",time)
printf("%-6s",status)
printf("%s\n", alltime)
}' $HOME/measure/ps.dat
```

 The procedure only works if the variable \$5 queries the current time zone. The time zones GMT, MET and MST are recognized. If necessary, you may have to modify the corresponding line accordingly.

Start the procedure as follows:

```
# ps_time_eval
```

Example output:

PID	Process name	SIZE	RSS	TIME	STAT	Time
1	/sbin/init	44	21	4:18	S	12:14:55
3	fsflush	0	0	47:20	S	12:14:55
1274	/usr/sbin/in.rwhod	281	62	2:05	S	12:14:55
15359	dw.sapC11_DVEBMGS00	17256	3010	2:04	R	12:14:55
15369	oracleC11	11077	4223	1:14	S	12:14:55
15358	dw.sapC11_DVEBMGS00	17258	3435	1:58	S	12:14:55
15368	oracleC11	11069	4028	1:09	S	12:14:55
1	/sbin/init	44	21	4:18	S	12:15:25
3	fsflush	0	0	47:20	S	12:15:25
1274	/usr/sbin/in.rwhod	281	62	2:05	S	12:15:25
15359	dw.sapC11_DVEBMGS00	17256	3008	2:23	R	12:15:25
15369	oracleC11	11077	4218	1:25	S	12:15:25
15358	dw.sapC11_DVEBMGS00	17258	3432	2:16	R	12:15:25
15368	oracleC11	11069	4021	1:20	S	12:15:25

The following columns are of importance for evaluation:

- SIZE
Virtual size of the process in Kbytes
- RSS
Resident Set Size
Size of the real main memory currently occupied by the process in Kbytes
Pages which are simultaneously used by multiple processes (e.g. shared memory pages, shared text pages) are divided equally between the individual processes.
- TIME
Total execution time in minutes and seconds

This record of CPU time utilization enables you to determine exactly which processes are placing an exceptional load on the processors.

If the CPU time utilization of a given process grows rapidly then under certain circumstances this may be caused by an infinite loop. Alternatively, the program may be waiting for an event which never occurs. You can check for this status by using, for example, the *truss(1)* command.

If you are certain that a process is not demanding excessive CPU time on account of an infinite loop or similar source of error but is genuinely extremely time-intensive, you may be able to improve the performance of the process by means of process scheduling (see section “Process scheduling” on page 134).

5.4 Analyzing the main memory resource

Main memory is a resource which greatly affects system performance. UNIX manages it in a way which provides optimum, high-performance memory use thanks to a special memory concept, so-called virtual memory (VM).

This section describes how you can analyze main memory load. It explains the dimensions which are relevant to this analysis and how they should be interpreted. This information can help you evaluate whether the main memory may represent a performance bottleneck for your system and your particular applications.

When analyzing the main memory you should consider the following factors:

- Characteristic main memory sizes (page 92)
- Size of free memory (page 93)
- System table utilization (page 95)
- Buffer cache load (page 97)
- Kernel memory allocation (page 98)
- Paging activity (page 100)
- Swapping activity (page 104)
- Process memory requirements (page 106)
- Analyzing the swap area (page 109)

You may use the following commands for analysis purposes:

- *sar(1)*

- *vmstat*(1M)
- *mpstat*(1M)

5.4.1 Characteristic main memory sizes

The size of the main memory is described in terms of the following characteristic sizes:

Total real memory

This value is the same as the size of the physical memory (possibly minus internal used memory pages by the hardware).

Available memory

This section of main memory is available following system start. It can be used both by the operating system and by applications.

The difference between Total real memory and Available memory is the part of main memory which is permanently occupied by the kernel. This memory section contains kernel code and statically configured system tables.

During system start, the characteristic sizes are displayed at the console and logged in the files `/var/adm/log/messages` and `/var/adm/log/osm`. If UNIX is running you can display these three characteristic values as follows:

```
# egrep mem /var/adm/log/osm
```

Example output for an RM600 E20:

```
Physical memory size = 1216 MB (77824 pages, pagesize=16 KB)
Total real memory   = 1275068416 B, (1216 MB)
Available memory    = 1202700288 B, (1146 MB)
                    1245184 KB of main memory
```

The following kernel parameters influence the way in which the kernel occupies main memory:

KV_SIZE, NPROC, BUFHWM, SM_SIZE

Some kernel parameters can influence main memory management because of the fact that the size of the kernel is dependent on them, e.g.:

UFSNINODE, NINODE, NCSIZE

5.4.2 Size of free memory

Virtual memory management uses the page as its management unit. A page is normally 16 Kbytes in size, in older SINIX versions 4 Kbytes. You should therefore first check the page size used by your system. The `pagesize(1)` command returns the value in bytes:

```
# /usr/ucb/pagsize
```

Example output:

```
16384
```

You should then check the number of available memory pages present. You may also evaluate the file `$HOME/measure/sar_bin.dat` which is generated by the procedure `main_meas` (see page 74):

```
# sar -r -f $HOME/measure/sar_bin.dat
```

Example output:

ReliantUNIX-Y wait 5.45 AI010 RM600 01/04/99			
00:00:01	freemem	freeswp	shmem
01:00:01	14035	923728	0
	.	.	.
07:00:00	19919	932976	0
08:00:00	19749	930296	0
09:00:00	14643	904240	0
10:00:00	9451	896632	0
	.	.	.
	.	.	.
Average	16767	914181	0

The meaning of the columns is as follows:

freemem

Average number of unused pages

These pages are available for processes or memory pools (see page 98).

If `freemem` falls below the value of `LOTSFREE` (in this example: 384 Kbytes), the `pageout` daemon starts to remove pages from main memory. In the example the size of the available memory at 10:00:00 was 9451 pages, that is to say approximately 147.7 Mbytes. Values which are marginally higher than the `LOTSFREE` specification indicate optimum main memory utilization.

freeswap

Average number of free blocks of 512 bytes in the swap area

For each newly started process it is necessary to reserve memory space in the swap area. This space must correspond to the data requirements of the process. If `freeswap` is very small (e.g. < 4 Mbytes) then it may not be possible to start new processes.

shmem

This parameter is not supported by the current version of Reliant UNIX.

Evaluations using vmstat

You can use the `vmstat(1M)` command to evaluate the allocation of virtual memory. This command was activated by the example procedure `vm_meas` (see page 72) and you are now able to evaluate the file `vm.dat`. The following example procedure `vm_mem_ausw` lists the columns `avm` and `fre`:

Procedure vm_mem_eval

```
# Procedure: vm_mem_eval

awk ' BEGIN {   printf("avm          ")
                printf("fre          ")
                printf("Time\n\n") }
      NR==1 {   intervall=$1
                next }
      NR==2 {   var=$4
                stu=substr(var,1,2)
                min=substr(var,4,2)
                sek=substr(var,7,2)
                next }
      $1=="procs" { next }
      $1=="r" { next }
      sek>=60 { sek=sek-60
                min=min+1 }
      min>=60 { min=min-60
                stu=stu+1 }
                { printf("%-10s", $4)
                  printf("%-8s", $5)
                  printf("%02d:%02d:%02d\n",stu,min,sek)
                  sek=sek+intervall
                } ' $HOME/measure/vm.dat
```

Start the procedure as follows:

```
# vm_mem_eval
```

Example output:

avm	fre	Time
		12:00:55
5032	427304	12:01:05
4928	426652	12:01:15
3820	426616	12:01:25
11736	426604	12:01:35
11940	426588	12:01:45
4880	426576	12:01:55
4956	426564	12:02:05
4100	426548	12:02:15
3820	426536	12:02:25
5700	426524	12:02:35
.	.	
.	.	

The columns have the following meanings:

avm
Number of active virtual pages of 1 Kbyte

fre
Size of the memory freelist in blocks of 1 Kbyte

For further information on *vmstat*, refer to section 3.6 on page 43 and section 5.4.6 on page 100.

5.4.3 System table utilization

The system tables are located in the kernel's virtual address space. The memory space required for these tables is dynamically reserved and released via kernel memory allocations (see page 98). They are not subject to paging operations.

The *sar* option *-v* provides the following information for important kernel tables: number of table entries currently occupied and maximum possible number of table entries. You may also evaluate the file *\$HOME/measure/sar_bin.dat* which is generated by the procedure *main_meas* (see page 74):

```
# sar -v -f $HOME/measure/sar_bin.dat
```

Example output:

ReliantUNIX-Y wait 5.44 B0021 RM600 01/04/98							
00:00:01	proc-sz	ov	inod-sz	ov	file-sz	ov	lock-sz
01:00:01	233/4116	0	148/7486	0	595	0	12/7502
	.	.	.	.	.	.	.
07:00:00	234/4116	0	206/7486	0	598	0	12/7502
08:00:00	232/4116	0	222/7486	0	597	0	13/7502
09:00:00	303/4116	0	204/7486	0	732	0	18/7502
10:00:00	314/4116	0	265/7486	0	763	0	17/7502
	.	.	.	.	.	.	.
	.	.	.	.	.	.	.
Average	270/4116	0	238/7486	0	667	0	15/7502

The meaning of the columns is as follows (numbers in all cases):

- proc-sz

Entries in the process table: occupied/maximum
- inod-sz

Entries in the UFS i-list (inode table): occupied/maximum
- file-sz

All files opened in system
- lock-sz

Entries in file/record lock table: occupied/maximum
- ov

Overflows

These values make it possible to determine whether the configured maximum values for the tables are sufficiently large. In the example above, only a fraction of the available resources are actually in use.

For further information on analyzing the system tables, refer to the section of the same name on page 150.

5.4.4 Buffer cache load

The (old) buffer cache is considered here because it may account for up to 5% of the real memory. If there is a memory bottleneck you can free up memory by reducing the size of the buffer cache to up to 1% of the real memory. However, this is only of use if the hit rate of the buffer cache is not impaired.

You can use the *sar* option *-b* to measure and display the hit rate. You may also evaluate the file *\$HOME/measure/sar_bin.dat* which is generated by the procedure *main_meas* (see page 74):

```
# sar -b -f $HOME/measure/sar_bin.dat
```

Example output:

ReliantUNIX-Y wait 5.45 A1010 RM600 01/04/99									
00:00:01	bread/s	lread/s	%rcache	bwrit/s	lwrit/s	%wcache	pread/s	pwrit/s	
01:00:01	0	3	76	0	1	3	0	0	
	.	.	.	.	.	.	.	.	
07:00:00	0	1	99	0	0	40	0	0	
08:00:00	0	1	93	0	0	44	0	0	
09:00:00	0	1	95	0	0	77	0	0	
10:00:00	0	1	99	0	0	78	0	0	
	.	.	.	.	.	.	.	.	
	.	.	.	.	.	.	.	.	
Average	0	9	94	2	4	68	0	0	

The following columns are of importance for evaluation:

%rcache

Percentage of hits in buffer cache on read accesses

%wcache

Percentage of hits in buffer cache on write accesses

i

If the average value of %rcache is < 90 or if that of %wcache is < 60 then it may be advisable to enlarge the buffer cache (kernel parameter *BUFHWM*). However, you should note that the buffer cache is only used for inodes, indirect blocks and other file system management information (see section “seg_kmem segment” on page 11).

In the example above the hit rate for *write* accesses at 01:00:01 is low. This is due to the particular type of access (data storage).

5.4.5 Kernel memory allocation

All operating system components are dynamically allocated the memory space they require via the kernel memory allocation facility and its memory pools (KMA = Kernel Memory Allocation).

- Memory allocation ≤ 16 Kbyte (i. e. $\leq$ page size)

The memory space allocation is unmapped, i.e. physical addresses are used directly in this case.

- Memory allocation > 16 Kbytes

Virtual addresses are used in this case. The total virtual address space available in the system kernel can be expanded to a maximum value which is defined by the *KV_SIZE* kernel parameter.

There are four different classes of KMA allocations:

1. Memory allocations from pools for small memory requests (1 - 512 bytes).
A pool is 16 Kbytes in size.
2. Memory allocations from pools for large memory requests (513 - 8 Kbytes).
A pool is 16 Kbytes in size.
3. Direct memory allocation from pages for oversized memory requests (> 8 Kbytes).
4. Allocation of memory which will never be returned to the operating system.
The memory is allocated on the basis of the size of requests from class 1, 2 or 3. A separate pool is provided for each group of requests. The size of each pool is variable.

The *sar -K* option provides information about the size of the aforementioned pools. You may also evaluate the *\$HOME/measure/sar_bin.dat* file, which was generated by the *main_meas* procedure (see page 74):

```
# sar -k -f $HOME/measure/sar_bin.dat
```

Sample output for `sar -K`:

ReliantUNIX-Y wait 5.45 A1010 RM600 01/04/99										
00:00:01	sm1_mem	alloc	fail	pool	lg_mem	alloc	fail	pool		
	fst_mem	alloc	fail	pool	ovsz_mem	alloc	fail			
00:00:01	2886912	2091616	0	0	179	17891328	16432640	0	1092	
	761424		0	0	10	406159360	406077440		0	
	.	.	.	.	.	.	.	.	.	.
07:00:00	2886912	2077664	0	0	179	17891328	15914496	0	1092	
	761424		0	0	10	406227118	406288563		0	
08:00:00	2957347	2063168	0	0	179	17891328	15557120	0	1134	
	801765		0	0	10	406159360	406077440		0	
	.	.	.	.	.	.	.	.	.	.
Average	2902636	2080034	0	0	179	18088208	15592268	0	1104	
	793411		0	0	11	406211392	406185984		0	

The meaning of the columns is as follows:

sm1_mem
Memory space in bytes, reserved and allocated for small pools

lg_mem
Memory space in bytes, reserved and allocated for big pools

fst_mem
Memory space in bytes, reserved and allocated for fast pools

ovsz_mem
Memory space in bytes, reserved and allocated for oversized pools

The columns, which are listed under sm1_mem, lg_mem, fst_mem and ovsz_mem, have the following meaning:

alloc
Number of bytes allocated for small, big, fast or oversized pools

fail
Number of failed requests per interval

pool
Number of small, big or fast pools

There is no pool for oversized memory space since it is reserved and allocated dynamically.

5.4.6 Paging activity

Paging occurs when main memory is withdrawn from a process and some of its pages are removed from memory. Paging is performed whenever executing processes require more free memory than is available.

If the size of the free memory falls below a limit value (*LOTSFREE*), then the *pageout* daemon is activated. This checks a particular number of main memory pages to determine whether they have remained unused for an extended period. If necessary, such pages are paged out to the swap area and identified as available.

If a process accesses an address whose associated page is not currently available in main memory, then this page is once again read into main memory from the hard disk (paging-in) where it may be stored either in the swap area or in a program file.

In terms of main memory allocation, an application is always running optimally when the system does not have to page out any data. The aim should be that the *pageout* daemon is never active or active for only very short periods during normal operation.

If data frequently has to be paged out then system performance will suffer. This is particularly serious when the *pageout* daemon has to page out data which then has to be immediately paged in again. In such cases, increasing the size of the physical memory can lead to significant improvements in performance.

Page-out activity

The *sar* option *-g* provides information about page-out activity. You may also evaluate the file *\$HOME/measure/sar_bin.dat* which is generated by the procedure *main_meas* (see page 74):

```
# sar -g -f $HOME/measure/sar_bin.dat
```

Example output:

ReliantUNIX-Y wait 5.45 a1010 RM600 01/04/99					
00:00:01	pgout/s	pggout/s	pgfree/s	pgscan/s	%ufsipf
01:00:01	0.01	0.01	0.01	0.00	0.00
	.	.	.	.	.
07:00:00	0.00	0.00	0.00	0.00	0.00
08:00:00	0.06	0.15	0.14	0.00	0.00
09:00:00	1.22	3.07	2.60	0.00	0.00
10:00:00	0.96	3.24	2.95	0.10	0.00
	.	.	.	.	.
12:20:55	0.00	0.00	0.00	0.00	0.00
Average	0.26	0.68	0.67	0.05	0.00

The columns have the following meaning:

- pgout/s

Number of page-out requests per second

Ideally this value should be zero.
- ppgout/s

Number of pages paged out per second

A single page-out request normally results in multiple pages being paged out. Consequently, ppgout/s is larger than pgout/s.
- pgfree/s

Number of pages released by the *pageout* daemon per second

The daemon also releases pages which are still required but have not been used for an extended period.
- pgscan/s

Number of pages checked by the *pageout* daemon per second.

High values (>15) indicate a high level of *pageout* daemon activity.
- %ufsipf

Percentage of *iget* calls which cause data to be paged out.

An *iget* call occurs when a file in the UFS (UNIX File System) is determined via an inode entry. If the value of %ufsipf is greater than 10%, you should increase the *UFSINODE* setting (see also section “Modifying i-lists (inode tables)” on page 150).

These statistics provide telling information as to whether enough main memory is present or whether a memory bottleneck is occurring. If the value of `ppgout/s` is much greater than 10 over an extended period, the system is suffering from a memory bottleneck.

Page-in activity

Ideally, when a virtual address is accessed, the corresponding page is directly available in main memory. For example, this may occur when there is sufficient space for all the code and data segments of all running processes in main memory.

The following situation is referred to as a page fault: The system attempts to access a virtual address to which no physical page has as yet been allocated. The page must therefore be provided. This can be done in a number of ways, some of which may be more “expensive” than others:

- The page may be detected in main memory.
- The page may be provided by copying another page which is present in main memory.
- Provision of the page may already have been initiated by another process.
- The page must be read from the hard disk.

The `sar` option `-p` provides information about paging-in activity resulting from the accessing of virtual addresses. You may also evaluate the file `$HOME/measure/sar_bin.dat` file which is created by the procedure `main_meas` (see page 74):

```
# sar -p -f $HOME/measure/sar_bin.dat
```

Example output:

ReliantUNIX-Y	wait	5.45	A1010	RM600		01/04/99
00:00:01	atch/s	pgin/s	ppgin/s	pflt/s	vflt/s	slock/s
01:00:01	15.68	10.96	33.61	1.73	12.50	6.13
	.	.	.	.	.	.
07:00:00	0.49	0.00	0.00	0.41	0.41	0.03
08:00:00	66.68	0.42	0.66	129.41	52.62	9.06
09:00:00	315.90	11.32	14.74	376.00	305.07	26.03
10:00:00	124.30	9.98	20.33	59.85	177.09	4.57
	.	.	.	.	.	.
12:20:55	0.10	0.00	0.00	0.00	0.10	0.20
Average	49.54	4.57	10.36	59.06	51.14	7.76

The columns have the following meanings (in all cases, number per second):

`atch/s`

Page faults involving pages which are already available in main memory

These pages are either entered in the memory freelist or are shared with other processes.

`pgin/s`

Page-in requests

`ppgin/s`

Pages paged-in

Paging-in means that the data has to be read from the hard disk. A single page-in request normally results in multiple pages being paged in.

Consequently, `ppgin/s` is larger than `pgin/s`.

`pflt/s`

Primarily: The number of pages which have to be duplicated because the data they contain requires modification. This operation, which is known as “copy on write” is very CPU-intensive.

Also: Number of accesses to illegal virtual addresses (protection faults).

`vflt/s`

Page faults resulting from the conversion of virtual to physical addresses

The physical addresses are generated from the virtual addresses by means of tables (page tables). `vflt/s` specifies the number of times that these tables have to be updated in main memory.

`slock/s`

Page errors relating to locked errors.

Pages may be locked, for example, while I/O operations are being performed between the hard disk and these pages.

These values do not provided a definitive indication of whether or not a memory bottleneck exists. They do, however, provide additional information about operating system activity resulting from the virtual memory concept.

The fact that a large number of pages are being paged in each second or that there is a high level of page faults (`vflt/s`) does not necessarily mean that there is a memory bottleneck since this mechanism is used for both program loading and data access.

High `vflt/s` values (> 100) result from attempts to access pages which have previously been paged out or are due to the fact that the required page has not yet been paged in (e g. on starting a new program).

Low values of `pgin/s` and `vflt/s` mean that precisely those pages that are relevant are located in the main memory.

Evaluation using `vmstat`

You can use the `vmstat(1M)` command to analyze paging activities:

```
# /usr/ucb/vmstat 5 5
```

Example output:

procs			memory		page						
r	b	w	avm	fre	re	at	pi	po	fr	sr	...
5	2	0	73288	46728	0	33	0	0	0	0	...
5	1	0	76176	47264	0	439	24	24	24	0	...
6	1	0	76520	46100	0	402	164	24	24	0	...
5	1	0	92008	45808	2	391	140	92	84	0	...
7	1	0	95136	45580	2	376	256	116	108	0	...

The following columns are of significance for analysis (in all cases, number per second):

- pi Pages paged-in
- po Pages paged-out
- fr Released pages
- sr Pages checked by the *pageout* daemon

For further information on `vmstat` refer to section “`vmstat(1M)`” on page 43 and section “Size of free memory” on page 93.

5.4.7 Swapping activity

The term swapping is used when complete processes are transferred out to or moved back in from the swap area. Swapping is performed when the available memory falls below a value which is defined by the kernel parameter `GPGSLO`. For example, this may occur when new memory-hungry processes are loaded even though the main memory load is already at a high level.

Swapping-out affects an entire process whose pages are then returned to the free memory management. Swapping-in describes the retrieval of previously swapped-out process when sufficient memory again becomes available.

Evaluation using sar

The `sar` option `-w` provides information about swapping-in and swapping-out activity. You may also evaluate the file `$HOME/measure/sar_bin.dat` which is generated by the procedure `main_meas` (see page 74):

```
# sar -w -f $HOME/measure/sar_bin.dat
```

Example output:

ReliantUNIX-Y	wait	5.45	A1010	RM600	01/04/99
00:00:01	swpin/s	pswin/s	swpot/s	pswot/s	pswch/s
01:00:01	0.00	0.0	0.00	0.0	23.9
	.	.	.	.	.
07:00:00	0.00	0.0	0.00	0.0	0.8
08:00:00	0.00	0.0	0.00	0.0	1.0
09:00:00	0.00	0.0	0.00	0.0	50.7
10:00:00	0.00	0.0	0.00	0.0	18.4
	.	.	.	.	.
12:20:55	0.00	0.0	0.00	0.0	0.7
Average	0.00	0.0	0.00	0.2	35.4

The columns have the following meanings (in all cases, number per second):

- swpin/s

Processes swapped-in (from the swap area to main memory)

When processes are swapped back in, only 12 Kbytes of process management data are swapped in. Any additional pages required are not paged in until they are accessed.
- pswin/s

Paged-in pages
- swpot/s

Swapped-out processes
- pswot/s

Paged-out pages

Normally, the last two values should be significantly smaller than 1. If this is not the case, then there is a memory bottleneck.
- pswch/s

Process switches

This operation is very CPU-intensive.

Evaluation using mpstat

The *mpstat*(1M) command provides a variety of statistics concerning current multiprocessor operation. Some of these statistics are listed separately for each processor. Some of the specifications relating to main memory and page transfer are similar to those provided by *vmstat*(1M) (see page 104).

```
# /usr/ucb/mpstat 5 5
```

Example output:

procs			memory		page				...
r	b	w	avm	fre	pi	po	si	so	
8	3	0	104624	31168	1	1	1	1	...
					1	1	1	1	...
10	0	0	77692	32092	13	0	0	0	...
					154	75	0	0	...
11	1	0	73576	36500	0	4	0	0	...
					165	39	0	0	...
5	2	0	80384	43068	1	0	0	0	...
					128	11	0	0	...
10	0	0	81188	45720	5	0	0	0	...
					160	0	0	0	...

The following columns are of significance for evaluation (in all cases, number per second):

- si Pages swapped in
- so Pages swapped out

For further information on *mpstat* refer to section “*mpstat*(1M)” on page 40.

5.4.8 Process memory requirements

If you think that memory bottlenecks are caused by the fact that individual processes are occupying too much memory space, you should analyze the memory requirements of these processes in greater detail. To do this You can use the *proceso*(1M) command (see page 54). You may also evaluate the file *\$HOME/measure/ps.dat* which is generated by the procedure *ps_meas* (see page 71).

The following procedure *ps_size_eval* lists all the processes whose virtual memory allocation (*SIZE*) is greater than or equal to 500 blocks.

Procedure ps_size_eval

```
# Procedure: ps_size_eval

awk ' BEGIN {   printf("PID      ")
                printf("Process name          ")
                printf("SIZE      ")
                printf("RSS       ")
                printf("TIME      ")
                printf("STAT      ")
                printf("Time\n\n") }
      $5=="GMT" || $5=="MET" || $5=="MST" { alltime=$4
                printf("\n") }
      $2=="Z" || $5=="SIZE" || $5=="GMT" || $5=="MET" || $5=="MST" { next }
      {glen=length($4)
      tlen=index($4,":")}
      tlen == 2 { time=substr($4, 1, 4) }
      tlen == 3 { time=substr($4, 1, 5) }
      {size=substr($4, tlen+3, glen)
      rss=$5
      pname=$6
      size==" " {size=$5
      rss=$6
      pname=$7}
      {status=$3
      pid=$1 }
      size>=500 { printf("%-7s", pid )
                  printf("%-30s", pname)
                  printf("%-8s", size)
                  printf("%-8s", rss)
                  printf("%-8s", time)
                  printf("%-6s", status)
                  printf("%s\n", alltime)
                }' $HOME/measure/ps.dat
```

i The procedure only works if the variable \$5 queries the current time zone. The time zones GMT, MET and MST are recognized. If necessary, you may have to modify the corresponding line accordingly.

Start the procedure as follows:

ps_size_eval

Example output:

PID	Process name	SIZE	RSS	TIME	STAT	Time
15375	oracleC11	10534	860	0:00	S	11:15:23
15323	ora_pmon_C11	10537	779	0:00	S	11:15:23
15361	dw.sapC11_DVEBMGS00	17133	996	0:00	S	11:15:23
15364	dw.sapC11_DVEBMGS00	17133	992	0:00	S	11:15:23
15327	ora_smon_C11	10575	962	0:00	S	11:15:23
15359	dw.sapC11_DVEBMGS00	17256	3008	2:23	R	11:15:23
15377	oracleC11	10619	1140	0:02	S	11:15:23
15374	oracleC11	10527	785	0:00	S	11:15:23
15369	oracleC11	11077	4218	1:25	S	11:15:23
15370	oracleC11	10527	784	0:00	S	11:15:23

The following columns are of significance for evaluation:

SIZE

Overall size of the process address space in Kbytes

SIZE specifies the virtual memory requirement. Even if this value is very large (e.g. 100 Mbytes), this does not mean that an equivalent amount of main memory is occupied. The sum of all the process address spaces may be greater than the physical main memory.

RSS

Resident Set Size

Size of the real main memory currently occupied by the process in Kbytes.

Pages which are simultaneously used by multiple processes (e.g. shared memory pages, shared text pages), are divided equally between the individual processes.

TIME

Total execution time in minutes and seconds

In this way you can determine the size of the virtual memory and the currently occupied physical memory for each individual process. You can use this memory allocation data to identify exactly which processes are occupying an exceptional amount of memory (e.g. > 10 Mbytes) and causing main memory bottlenecks.

If the amount of memory occupied by a particular process grows continuously, the application may contain a programming error, e.g. it may constantly occupy new memory without subsequently releasing this again.

In the example given here, some of the ORACLE® database processes occupy up to 16 Mbytes of virtual main memory.



Depending on the implementation, it may happen that the RSS values of processes which have been swapped-out and then swapped-in is not correctly calculated if there is simultaneously a high level of paging activity. If such a situation arises, the values which are calculated for the processes in question are too small.

Example output for `ps -l`

F	S	UID	PID	PPID	C	PRI	NI	ADDR	SZ	WCHAN	TTY	TIME	CMD
10	S	0	1999	1998	1	82	20	c1202454	18	c11e3800	inet/59	0:00	sh
10	O	0	5661	1999	5	82	20	c11e3854	19		inet/59	0:00	ps

The meaning of the column `SZ` is as follows:

`SZ` Total size of the process address space in pages of 16 Kbytes

If you notice that the `SZ` values of a process grow constantly, the reason may be as follows: The process requests memory which it then, incorrectly, fails to release.

You can use the kernel parameters `SDATLIM`, `SSTKLIM` and `SVMMLIM` to prevent uncontrolled growth of the process address space (see page 198).

5.4.9 Analyzing the swap area

By its nature, the swap area resource more properly belongs to the section “Hard disks and file systems”. However, as the use of this resource is highly dependent on the main memory load, it is also discussed here.

The necessity of a swap area is a consequence of the virtual memory concept. The swap area is used for the temporary storage of pages which have to be removed from main memory. The size of the swap area (and not the size of the available main memory) determines the maximum size of the address spaces of all the processes which are simultaneously active in the system.

The way in which you ascertain whether and how frequently memory bottlenecks occur as a result of swapping activity in the system has already been described on page 104.

Determining the size of the swap area

swap -l

Example output:

path	dev	swaplo	blocks	free
/dev/vd/vdisk1	7,1	0	819200	803200
/dev/ios0/sdisk011s13	4,2189	0	2059264	2044928

The meaning of the columns is as follows:

blocks
 Partition size in blocks of 512 bytes

free
 Number of free blocks within the partition

In the example, the swap area consists of 2 hard disk partitions. It has a total size of approximately 562 Mbytes. Of these, approximately 415 Mbytes are free.

When a process is started or if the process requests additional main memory, then sufficient memory is reserved in the swap area for the process to be fully swapped out to the swap area in the event of a main memory bottleneck.

If there is no longer sufficient memory available in the swap area, it is not possible to start a new process: Sorry, pid was killed due lack of swap space. The configured value of the swap area is too small. If the swap area is even smaller than the main memory, a message to this effect is output on the console at system start. For information on enlarging the swap area, refer to the section “Increasing the swap area” on page 137 and the *swap(1M)* manual page.

Determining the current allocation

swap -s

Example output:

```
total: 147088 allocated + 169088 reserved = 316176 blocks used, 768152 blocks available
```

The meaning of the columns is as follows:

blocks available
Free blocks of 512 bytes available for users

blocks used
Allocated blocks of 512 bytes (total of allocated and reserved)

5.5 Hard disks and file systems

This section describes how you can analyze the utilization of hard disks and file systems.

5.5.1 Identifying installed hard disks and SCSI controllers

You can use the *autoconf*(8) command to display all the connected hard disks and SCSI controllers which are known to the system:

```
# autoconf -l | grep sdisk
```

Example output for an RM600 E:

```
ios0/sdisk011 HD11 -- HD11
ios0/sdisk012 HD04 -- HD04
ios0/sdisk013 HD04 -- HD04
ios0/sdisk014 HD04 -- HD04
ios0/sdisk022 HD28 -- HD28
ios0/sdisk023 HD38 -- HD38
ios0/sdisk024 HD27 -- HD27
```

The above example lists SCSI hard disks of different types with capacities of between 4 Gbytes and 34 Gbytes.

5.5.2 Displaying/modifying the partition layout of a hard disk

The current division of a hard disk into partitions (slices) and the size of these partitions is entered in a partition table. The partition table is stored in a special sector of the hard disk which is not accessible to users.

You can use the `dkpart(8)` command to display this hard disk layout. For example:

```
# dkpart -lb /dev/ios0/rsdisk012
```

Example output:

ios0/sdisk012 "HD04: 5.45 " Geometry 17356-3:16:64 (cyls:heads:sectors)
Sector size = 512 bytes 8678.0 Total Mbytes (1MB = 1024 * 1024 bytes)

Partition	Start	Rule	End	Rule	First Blk	Last Blk	Blk Count	Size (MB)
0	8		512		8192	532479	143360	256.0
1		>p0	2048		532480	2629631	262144	1024.0
2		>p1	1024		2629632	3678207	409600	512.0
3		>p2	1024		3678208	4726783	348160	512.0
4		>p3	1024		4726784	5775359	204800	512.0
5		>p4	3520		5775360	9379839	2747392	1760.0
6		>p5	8192		9379840	17768447	4119552	4096.0
7	0		0			* * *	Unused	* * *
8	0		0			* * *	Unused	* * *
9	0		0			* * *	Unused	* * *
10	0		8		0	8191	8192	4.0
11	0		0			* * *	Unused	* * *
12	0		0			* * *	Unused	* * *
13	0		0			* * *	Unused	* * *
14	0		0			* * *	Unused	* * *
15		>p6	\$		17768448	17769471	1024	0.5

Partition layout (not to scale):
|10|-0-|---1---|---2---|---3---|---4---|-----5-----|-----6-----|15|

The `dkpart -e` command allows you to modify the partition layout. These modifications take effect immediately without you having to install a new kernel.



When you modify the partition layout, you can no longer access any file systems which may be located in these partitions. All the data contained in these file systems is lost!

You can also modify the layout of a hard disk by setting up virtual disks, for example via partition 7 (see page 138).

5.5.3 Identifying partitions used as file systems

You can use the *dkmap*(8) to display the purpose for which each partition is being used. If some file systems are not mounted, then the *df*(1) command also fulfils this function. For example:

```
# dkmap
```

Example output:

```
ios0/sdisk011      (4023.5 Mbytes):
    /dev/ios0/sdisk011s0      ufs:/
    /dev/ios0/sdisk011s1      AVAILABLE (262144 b1ks)
    /dev/ios0/sdisk011s2      vxfs:/opt
    /dev/ios0/sdisk011s3      vxfs:/usr
    /dev/ios0/sdisk011s4      vxfs:/var
    /dev/ios0/sdisk011s5      vxfs:/home
    /dev/ios0/sdisk011s8      AVAILABLE (4119552 b1ks)
    /dev/ios0/sdisk011s10     <raw>
    /dev/ios0/sdisk011s15     AVAILABLE (1024 b1ks)
ios0/sdisk012      (4023.5 Mbytes):
    /dev/ios0/sdisk012s7      AVAILABLE (8239104 b1ks)
    /dev/ios0/sdisk012s15     AVAILABLE (1024 b1ks)
ios0/sdisk013      (4023.5 Mbytes):
    /dev/ios0/sdisk013s7      AVAILABLE (8239104 b1ks)
    /dev/ios0/sdisk013s15     AVAILABLE (1024 b1ks)
ios0/sdisk014      (4023.5 Mbytes):
    /dev/ios0/sdisk014s7      AVAILABLE (8239104 b1ks)
    /dev/ios0/sdisk014s15     AVAILABLE (1024 b1ks)
ios0/sdisk015      (4023.5 Mbytes):
    /dev/ios0/sdisk015s6      vxfs:/dumps
    /dev/ios0/sdisk015s8      <swap>
    /dev/ios0/sdisk015s15     AVAILABLE (1024 b1ks)
ios0/sdisk016      (4023.5 Mbytes):
    /dev/ios0/sdisk016s7      AVAILABLE (8239104 b1ks)
    /dev/ios0/sdisk016s15     AVAILABLE (1024 b1ks)
ios0/sdisk021      (4023.5 Mbytes):
    /dev/ios0/sdisk021s7      AVAILABLE (8239104 b1ks)
    /dev/ios0/sdisk021s15     AVAILABLE (1024 b1ks)
    .
    .
```

The device filenames tell you which file system is located on which partition.

The device filenames are explained in the *sdisk*(7) manual page.

5.5.4 Determining the fill level of file systems

File systems should only be occupied to a maximum level of 80%. Beyond this level it becomes increasingly difficult for the system to find free blocks located in the same cylinder group.

There are two commands available for determining the extent to which disk capacity is utilized by file systems: *df(1)* and *dfspace(1)* with a direct specification of Mbytes and percentage. If partitions are used as swap areas, you can use the *swap -s* and *swap -l* commands to determine which swap areas are in use or are currently occupied (see page 109). If partitions are managed by applications themselves independently of the operating system, then you should evaluate these using the appropriate commands or tools (e.g. using *onmonitor* for INFORMIX-ONLINE® or *sqldb* for ORACLE).

The example below illustrates how you can display disk allocation by file system (provided that the file systems are mounted):

```
# df -k
```

Example output:

Dateisystem	Kbytes	belegt	verf.	Kapazität	eingehängt auf
/dev/root	39306	20540	14834	58%	/
/proc	0	0	0	0%	/proc
/dev/fd	0	0	0	0%	/dev/fd
/dev/ios0/sdisk000s2	92874	81236	2350	97%	/opt
/dev/ios0/sdisk000s3	127026	99996	14326	87%	/usr
/dev/ios0/sdisk000s4	68286	11090	50366	18%	/var
/dev/ios0/sdisk000s5	179814	18330	143502	11%	/home
/dev/vd/vdisk0	174510	82	156976	0%	/frei
/dev/ios0/sdisk002s13	320664	222480	98184	69%	/usr/kunde/C11
/dev/ios0/sdisk002s12	237016	1112	235904	0%	/usr/sap
/dev/ios0/sdisk002s3	130824	52560	78264	40%	/usr/ora
/dev/vd/vdisk2	94176	82096	11136	88%	/usr/ora/C11/sapdat4
/dev/ios0/sdisk002s14	323656	205032	118624	63%	/usr/ora/C11/sapdat0
/dev/ios0/sdisk010s7	1294096	730784	563312	56%	/usr/ora/C11/sapdat1
/dev/ios0/sdisk011s7	1294096	1009384	271768	79%	/usr/ora/C11/sapdat3
/dev/ios0/sdisk012s8	647336	46152	601184	7%	/usr/ora/C11/sapreorg
/dev/ios0/sdisk012s12	239096	16	239080	0%	/usr/ora/C11/saparch

In this example, although the occupancy level of the */home* directory is low, that of the */opt* file system is running at 97%. In such cases it is advisable to look for very large file systems or file systems which have long remained unused and which may be superfluous and can be deleted or transferred to an archive. For more information, see the chapter “Implementing tuning measures” on page 133.

Another way of overcoming these capacity problems would be to reconfigure or enlarge the partition for the */opt* file system.

5.5.5 Checking whether the CPU is waiting for I/O

Analysis of the CPU load can tell you whether the CPU has been idle while waiting for input or output. The *sar* options *-P* (see page 84) and *-u* provide data about this. You may also evaluate the file *\$HOME/measure/sar_bin.dat* which is generated by the procedure *main_meas* (see page 74):

```
# sar -u -f $HOME/measure/sar_bin.dat
```

Example output:

ReliantUNIX-Y wait 5.45 A1010 RM600 01/04/99						
12:00:55	%usr	%sys	%wio	%idle	int/s	intdef/s
12:01:05	2	4	0	94	6	0
12:01:15	10	15	2	73	24	0
12:01:25	4	8	18	69	31	0
12:01:35	10	15	35	39	67	0
12:01:45	19	32	43	6	93	0
	.	.	.	.	.	.
	.	.	.	.	.	.
	.	.	.	.	.	.
12:20:45	2	3	0	95	9	0
12:20:55	0	0	0	100	5	0
Average	18	12	4	66	93	0

The columns *%usr*, *%sys*, *%wio* and *%idle* must all be taken into account during evaluation.

%usr
Proportion of CPU time in user mode

%sys
Proportion of CPU time in system mode

Very high values (about 100%) mean that the user processes are performing large numbers of, or very intensive, system calls.

%wio
Idle; a process is waiting for block input or output

%idle
Idle; the CPU is waiting for processes to become executable

int/s
Number of interrupts per second

intdef/s
Number of delayed interrupts per second

In the current Reliant UNIX version this value is always 0.

Block devices consist of hard disks as well as remote file systems such as NFS and diskette drives. In this example, the CPU is spending up to 43% of runtime waiting for input from or output to block devices.



Simultaneous high values of `%idle` and `%wio` (e.g. > 80%) and poor response times indicate input/output problems or locking conflicts (semaphores).

5.5.6 Determining the disk load

The utilized file systems and partitions should be evenly distributed over the available disks. In doing this, you should not only take account of the partitions with UNIX file systems but also of those partitions which can be written or read directly via a device in raw mode (e.g. ORACLE and INFORMIX-ONLINE database areas). For example, a frequently used file system should not share a disk with a frequently used database area! The aim is to distribute the load evenly across all the disks.

A possible distribution for INFORMIX-SE[®] might be as follows: by default, INFORMIX-SE saves temporary tables in `/tmp`. The database should therefore not be located on the same disk as `/tmp`.

There are three ways of ascertaining hard disk load:

- measurement with *sar*
- using the *vmstat* statistics
- using the *dkstat* statistics (most detailed method!)

Evaluation using *sar*

You can use the *sar* option `-d` to evaluate the activity of all block devices. You may also evaluate the file `$HOME/measure/sar_bin.dat` which is generated by the procedure *main_meas* (see page 74):

```
# sar -d -f $HOME/measure/sar_bin.dat
```

Example output:

ReliantUNIX-Y wait 5.44 B0021 RM600 01/04/98							
12:00:55	device	%busy	avque	r+w/s	blks/s	avwait	avserv
12:01:05	ios0/sdisk000	0.92	1.70	1	7	13.20	18.40
12:01:15	ios0/sdisk000	1.22	1.02	0	7	0.50	30.50
	ios0/sdisk002	3.84	1.01	1	19	0.50	32.00
12:01:25	ios0/sdisk000	1.48	2.35	1	3	29.14	21.14
	ios0/sdisk010	8.15	1.54	3	46	15.34	28.10
	ios0/sdisk002	34.49	1.01	12	154	0.73	28.27
	ios0/sdisk011	9.04	1.56	3	46	18.00	31.17
	ios0/sdisk012	3.90	1.00	2	21	0.53	26.00
12:01:35	ios0/sdisk000	10.01	7.34	4	85	184.14	28.60
	ios0/sdisk010	8.88	3.03	3	53	55.09	26.91
	ios0/sdisk002	76.36	2.09	31	403	28.18	25.04
	ios0/sdisk011	8.09	2.75	3	48	47.37	26.97
	ios0/sdisk012	1.88	1.94	1	7	35.60	37.60
	.	.	.	.	.	.	.
	.	.	.	.	.	.	.
	.	.	.	.	.	.	.
12:20:45	ios0/sdisk000	2.26	1.71	1	11	23.14	32.29
12:20:55	ios0/sdisk000	1.35	1.31	0	5	14.33	45.00
Average	ios0/sdisk000	3.65	3.40	1	26	72.01	29.74
	ios0/sdisk010	3.19	1.37	1	17	10.89	28.17
	ios0/sdisk002	4.56	3.19	2	26	52.59	23.78
	ios0/sdisk011	3.14	1.43	1	17	12.89	28.71
	ios0/sdisk012	1.01	3.31	0	8	60.56	26.15

The meanings of the columns are as follows:

device

Name of the monitored disk

%busy

Proportion of time in which a block device was busy with the execution of a transfer job.

This value is critical if it is continuously greater than 60%.

avque

Average number of jobs in the job queue during the time specified in %busy.

`r+w/s`

Number of read/write operations per second

`blks/s`

Number of transferred blocks of 512 bytes

`await`

Average time in milliseconds which data transfer jobs spent in the queue

`avserv`

Average time in milliseconds taken to process data transfer jobs.

In the case of disks this includes the search operation, head positioning, disk spin-dependent delays and the time required for transfer.

In this example, the distribution of the utilized file systems would be good if approximately equal values were output for all the disks. The distribution of the utilized file systems would be poor if high values were output for one disk only. High values for one disk only indicate that this disk is subjected to a heavy load.

The load of the individual hard disks can only be analyzed over an extended period. The following values are of significance for performance: the number of read/write operations (column `r+w/s`) as well as the number of 512-byte blocks transferred (column `blks/s`). Normally, the `%busy` value of hard disks in continuous operation should not exceed 60% as higher values impair response times. Given today's access optimization procedures (e.g. look ahead mechanisms) the `%busy` value may even exceed 100% during periods of high load and sequential read accesses (see `sar(1)` in the "System Administrator's Reference Manual").

In the case of "random" disk access (e.g. update-intensive database processes), the `r+w` value (number of I/O operations) should also be considered. Here, the maximum possible number of I/O operations depends on the type of hard disk used and in the case of an MP81 has a value of approximately 60 I/Os per second. This value can only be increased by distributing the I/O load over multiple disks, for example by using virtual disks (see section "Selecting virtual disk management" on page 138).

In the case of sequential I/O (e.g. writing a large file, database log areas) the limit values are some 30% higher since the disk needs to perform fewer positioning operations than in the case of random access. In the case of sequential access, performance cannot be boosted. Here you should simply bear in mind that disk areas in which a large number of random I/Os occur should not be located on a disk in which high throughput is required from sequential I/Os in another partition.

To obtain information about I/O distribution over the individual partitions use the *dkstat(1M)* command (see section below).

Evaluation with dkstat

You can also use the *dkstat(1M)* command to analyze hard disk activity. This command was started by the example procedure *dk_meas* (see page 72), and you can now evaluate the file *dk.dat*. The example procedure *dk_disk_eval* below lists all the columns of the *dkstat* command (s000, ...):

Procedure dk_disk_eval

```
# Procedure: dk_disk_eval
awk '
    NR==1 { intervall=$1
            next }
    NR==2 { var=$4
            stu=substr(var,1,2)
            min=substr(var,4,2)
            sek=substr(var,7,2)
            next }
    $1=="Average" { next }
    $1=="s000" { printf ("%s    Time\n", $0)
                next }
    sek>=60 { sek=sek-60
              min=min+1 }
    min>=60 { min=min-60
              stu=stu+1 }
            { printf("%s", $0)
              printf("    %02d:%02d:%02d\n",stu,min,sek)
              sek=sek+intervall
            } ' $HOME/measure/dk.dat
```

Start the procedure as follows:

```
# dk_disk_eval
```

Example output:

0s000	0s001	0s010	0s002	0s011	0s012	Time
0	0	0	0	0	0	12:00:55
0	0	0	1	0	0	12:01:05
0	0	2	12	2	1	12:01:15
3	0	3	30	3	0	12:01:25
4	0	1	41	1	0	12:01:35
3	0	7	17	2	0	12:01:45
2	0	2	2	3	0	12:01:55
0	0	4	6	5	0	12:02:05
1	0	3	2	3	0	12:02:15
0	0	0	0	0	0	12:02:25
2	0	0	0	0	0	12:02:35
0	0	2	0	4	0	12:02:45
0	0	5	3	2	0	12:02:55
2	0	4	12	2	1	12:03:05
1	0	5	7	3	0	12:03:15
.	.	.	.	.	.	
.	.	.	.	.	.	

The meaning of the columns is follows:

0s0xx
Number of I/Os per second.

However, *dkstat* is far more powerful than might appear here! You can use this command to generate comprehensive disk statistics. For a detailed description of the command see page 38.

5.5.7 Caching file I/O

File I/O is performed via the *segmap* segment of virtual memory. In contrast, the (old) buffer cache is used for the file system metadata. For a detailed description of this operation, refer to the section “seg_kmem segment” on page 11.

You can use the *sar -b* command to determine the buffer cache load. You may also evaluate the file *\$HOME/measure/sar_bin.dat* which is generated by the procedure *main_meas* (see page 74):

```
# sar -b -f $HOME/measure/sar_bin.dat
```

Example output:

ReliantUNIX-Y wait 5.45 A1010 RM600 01/04/99								
12:00:55	bread/s	lread/s	%rcache	bwrit/s	lwrit/s	%wcache	pread/s	pwrit/s
12:01:05	0	3	100	0	1	92	0	0
12:01:15	0	2	95	0	2	73	0	0
12:01:25	0	53	99	5	10	49	0	0
	.	.	.	.	.	.	.	.
12:20:35	0	1	100	0	0	100	0	0
12:20:45	0	1	100	0	0	100	0	0
12:20:55	0	1	100	0	0	100	0	0
Average	0	9	100	2	4	57	0	0

Te meanings of the columns are as follows:

- bread/s
Average number of 512-byte blocks read from disk per second.
- lread/s
Average number of blocks read from the system buffer.
- %rcache
Percentage hits in buffer cache on read access.
- bwrit/s
Average number of 512-byte blocks written to disk per second.
- lwrit/s
Average number of blocks written to the system buffer.
- %wcache
Percentage hits in buffer cache on write access.
- pread/s
Average number of device read accesses in raw mode.
- pwrit/s
Average number of device write accesses in raw mode.



If the average value of %rcache is < 90 or if that of %wcache is < 60 then it may be advisable to enlarge the buffer cache (kernel parameter *BUFWHM*). However, you should note that the buffer cache is only used for inodes, indirect blocks and other file system management information (see section “seg_kmem segment” on page 11).

5.5.8 Determining file access frequencies

Before a file can be accessed, its inode must have been read from the disk into main memory (“inode cache”). This occurs when the file is first addressed under its pathname. At this point, its pathname must be resolved.

Each time a pathname is resolved, the operating system must determine the index number associated with each component of the pathname and search for the appropriate inode in the inode cache or, if it is not present, read it into this cache. This operation can be highly time-intensive if a large number of small directories have to be searched.

The *sar* option *-a* provides information about the activities involved in this operation. You may also evaluate the file *\$HOME/measure/sar_bin.dat* which is generated by the procedure *main_meas* (see page 74):

```
# sar -a -f $HOME/measure/sar_bin.dat
```

Example output:

ReliantUNIX-Y wait 5.45 A1010 RM600 01/04/99			
12:00:55	iget/s	namei/s	dirbk/s
12:01:05	1	10	0
12:01:15	5	26	0
12:01:25	57	72	2
12:01:35	14	56	1
12:01:45	13	39	1
12:01:55	101	142	4
	.	.	.
12:20:35	0	9	0
12:20:45	0	0	0
12:20:55	0	0	0
Average	3	11	0

The meanings of the columns are as follows (in all cases, number per second):

iget/s
 Searches for inodes in the inode cache

namei/s
 Pathname resolutions

dirbk/s
 Directory read operations
(corresponds to *readdir/s* in *sar -x* but has nothing to do with the resolution of pathnames)

When pathnames are resolved, each component is initially looked for in the DNLC (Directory Name Lookup Cache). If it is found, its inode does not have to be searched for in the inode cache.

`iget/s` is therefore only specified if a pathname component cannot be found in the DNLC. The `iget/s` to `namei/s` ratio consequently provides a measure of the hit rate in the DNLC. in general it should not exceed 1:4, i.e. in four resolutions of full pathnames the inode cache should be searched no more than once.

The size of the DNLC can be determined using the kernel parameter `NCSIZE` (see page 187). If you enlarge the DNLC you may also need to enlarge the inode cache since the inode cache must always be significantly larger than the DNLC.

The `sar` option `-x` provides information about UFS file system activity. You may also evaluate the file `$HOME/measure/sar_bin.dat` which is generated by the procedure `main_meas` (see page 74):

```
# sar -x -f $HOME/measure/sar_bin.dat
```

Example output:

ReliantUNIX-Y wait 5.44 B0021 RM600 01/04/98							
12:00:55	open/s	create/s	lookup/s	readdir/s	getpage/s	putpage/s	other/s
12:01:05	UFS						
12:01:15	82.61	0.00	708.70	71.01	39.13	421.74	1333.33
12:01:25	UFS						
12:01:35	22.00	0.00	218.00	18.00	9.00	56.00	324.00
12:01:45	UFS						
12:01:55	785.00	0.00	2266.00	14.00	225.00	86.00	3129.00
12:02:05	UFS						
12:02:15	32.73	0.00	312.73	30.91	12.73	227.27	630.00
12:02:25	UFS						
12:02:35	891.11	0.00	2446.67	6.67	268.89	35.56	3467.67
12:02:45	:						
12:02:55	:						
12:03:05	:						
Average	UFS						
12:03:15	288.81	0.00	935.69	36.34	93.84	83.84	1616.60

The meanings of the columns are as follows (in all cases, number per second):

- open/s
 - UFS file opened
- create/s
 - UFS file created
- lookup/s
 - Search for filenames in a UFS directory

- readdir/s
 - Read all entries in a UFS directory
(see system call *getdents(2)*)
 - getpage/s
 - Supply list of pages in a UFS file
 - Pages which are not found in main memory must be read from disk.
 - putpage/s
 - Write list of pages of a UFS file to disk
 - other/s
 - Other UFS file operations
e.g. determine/modify owners or access permissions, allocate new names etc.
- See also section “Reducing the time taken to search directories” on page 147.

5.5.9 Analyzing inode utilization in file systems

When examining file access frequencies, it is also of interest to know the number of inodes which are available per file system and how many of these are actually in use. This information is supplied by the following command:

```
# /usr/ucb/df -i
```

Example output:

Filesystem	iused	ifree	%iused	Mounted on
/dev/root	3439	14993	19%	/
/dev/ios0/sdisk035s3	11786	37366	24%	/usr
/dev/ios0/sdisk035s4	1369	47783	3%	/var
/dev/ios0/sdisk012s8	29	516067	0%	/dumps
/dev/ios0/sdisk011s5	40258	484030	8%	/home
/dev/ios0/sdisk035s2	7429	74491	9%	/opt
/dev/ios0/sdisk031s8	32	491488	0%	/work031s8
/dev/ios0/sdisk032s12	32	262112	0%	/work032s12
.	.	.	.	.
/dev/vd/vdisk1	12	98292	0%	/mdisk1
/dev/vd/vdisk2	12	98292	0%	/mdisk2
/dev/vd/vdisk3	12	98292	0%	/mdisk3
/dev/vd/vdisk4	12	98292	0%	/mdisk4
/dev/vd/vdisk5	12	98292	0%	/mdisk5

The meanings of the columns are as follows:

`iused`

Number of used inodes

`ifree`

Number of free inodes

`%iused`

Percentage of used inodes

It may happen that a file system no longer possesses sufficient inodes. In such cases, no new files can be created in the file system, even if the file system still possesses enough free blocks!

To increase the number of inodes, proceed as follows:

- ▶ Save the file system in question (logical safeguard!) (using *cpio* or *tar* but not, however, *dd*).
- ▶ Create a new file system which contains more inodes (UFS file system: with *newfs*(1M) or *mkfs*(1M), VERITAS file system: with *mkfs*(1M)).
- ▶ Then read the saved file system back in.

By default, *newfs* generates an inode for every 2 Kbytes of user data, assuming that the files have an average size of 2 Kbytes. You can change this value by specifying the *-i* option when you enter *newfs*. However, inode overflows are rare events and this measure is seldom needed.

5.5.10 Analyzing system calls

The system call statistics can provide important information. You may also evaluate the file `$HOME/measure/sar_bin.dat` which is generated by the procedure *main_meas* (see page 74):

```
# sar -c -f $HOME/measure/sar_bin.dat
```

Example output:

ReliantUNIX-Y wait 5.45 A1010 RM600 01/04/99							
12:00:55	scall/s	sread/s	swrit/s	fork/s	exec/s	rchar/s	wchar/s
12:01:05	127	59	2	0.70	1.00	81057	2922
12:01:15	307	28	20	1.20	1.20	8565	3548
12:01:25	215	33	13	0.20	0.20	87310	14357
	.	.	.	.	.	.	.
12:02:05	294	103	28	0.70	0.70	171570	23762
12:02:15	565	102	63	0.10	0.30	361915	30749
12:02:25	287	65	52	0.00	0.00	78915	7740
Average	532	197	173	0.31	0.32	66656	20640

The meaning of the columns is as follows (in all cases, number per second):

- scall/s
System calls
- sread/s
read system calls
- swrit/s
write system calls
- fork/s
fork system calls
- exec/s
exec system calls
- rchar/s
characters read
- wchar/s
characters written

In general, *read* and *write* accesses account for approximately half of all system calls although this proportion may vary greatly depending on the activity which the system is performing.

It is not generally a cause for concern if the number of system calls per second is very high since there are “cheap” and “expensive” system calls. However, in this case you should conduct a detailed analysis during system operation using the *scstat(8)* command. This logs the system calls which are handled at each processor and the length of time taken by the processors to process the individual system calls (see section “*scstat(8)*” on page 47).

5.6 System tables

5.6.1 Analyzing the process table, i-list (inode table) and file table

You can analyze all three tables using the *sar* option *-v*. You may also evaluate the file *\$HOME/measure/sar_bin.dat* which is generated by the procedure *main_meas* (see page 74):

```
# sar -v -f $HOME/measure/sar_bin.dat
```

This provides information relating to the level of process table use, the index entries (i-lists) and the file tables. The current sizes of the kernel tables and the associated maximum values are output.

Example output:

ReliantUNIX-Y wait 5.45 AI010 RM600 01/04/99							
12:00:55	proc-sz	ov	inod-sz	ov	file-sz	ov	lock-sz
12:00:55	75/8212	0	573/14825	0	239	0	8/14875
12:00:55	75/8212	0	573/14825	0	239	0	8/14875
12:00:55	75/8212	0	573/14825	0	239	0	8/14875
	:	:	:	:	:	:	:
13:19:11	75/8212	0	573/14825	0	239	0	8/14875
13:19:21	74/8212	0	573/14825	0	238	0	8/14875
13:19:31	72/8212	0	573/14825	0	234	0	8/14875
Average	75/8212	0	573/14825	0	238	0	8/14875

The meanings of the columns are as follows (number in all cases):

- proc-sz

Entries in the process table: used/maximum
- inod-sz

Entries in the UFS i-list (inode table): used/maximum
- file-sz

All files opened in system
- lock-sz

Entries in the table for files/record locks: used/maximum
- ov

Overflows

The above example shows that all the tables are sufficiently large to prevent any overflow.

5.6.2 Analyzing message queues

You can use the *sar* option *-m* to analyze message queue activity. You may also evaluate the file *\$HOME/measure/sar_bin.dat* which is generated by the procedure *main_meas* (see page 74):

```
# sar -m -f $HOME/measure/sar_bin.dat
```

Example output:

ReliantUNIX-Y wait 5.45 A1010 RM600 01/04/99		
12:00:55	msg/s	sema/s
12:01:05	0.00	0.00
	.	.
12:04:15	12.30	11.90
12:04:25	1.40	5.20
12:04:35	0.00	1.30
12:04:45	6.70	41.60
12:04:55	1.48	16.85
12:05:05	3.15	31.07
12:05:15	0.00	1.30
12:05:25	22.60	31.90
12:05:35	2.10	3.20
12:05:45	2.40	19.40
	.	.
12:20:55	0.00	0.00
Average	2.68	56.50

The meanings of the columns are as follows (in all cases, number per second):

- msg/s
 Message accesses
- sema/s
 Semaphore activity

Evaluation with ipcs

If the measuring procedure periodically starts the *ipcs -qa* command (as in the example procedure *ps_meas* on page 71), then the file *\$HOME/measure/ipcs_msg.dat* contains information about the message queues. This information tells you which and how many message queues have been opened together with their fill level (in bytes).

Example output:

```
IPC status from /dev/kmem as of Mon Jan 04 12:00:55 1999
T   ID      KEY      MODE      OWNER ..... CBYTES  QNUM QBYTES ...
Message Queues:
IPC status from /dev/kmem as of Mon Jan 04 12:01:26 1999
T   ID      KEY      MODE      OWNER ..... CBYTES  QNUM QBYTES ...
Message Queues:
IPC status from /dev/kmem as of Mon Jan 04 12:01:57 1999
T   ID      KEY      MODE      OWNER ..... CBYTES  QNUM QBYTES ...
Message Queues:
q   100 0x00009c41 --rw-rw-rw-  c1ladm .....      0      0 16384 ...
IPC status from /dev/kmem as of Mon Jan 04 12:02:28 1999
T   ID      KEY      MODE      OWNER ..... CBYTES  QNUM QBYTES ...
                                     :      :      :
Message Queues:                       :      :      :
```

The following columns are of significance for evaluation:

- CBYTES

The number of bytes in the messages waiting for processing in the corresponding queues.
- QNUM

The number of messages currently present in the corresponding message queue.
- QBYTES

The maximum total number of bytes permitted in messages waiting for processing in the message queue.
- This evaluation enables you check whether the configuration of the message queues needs to be changed in the kernel parameters. When using message queues you should ensure that you check the following kernel parameters against the measured values (see also page 193):
- MSGMNI

Maximum number of message queues. Each line of each measurement period logged in the file *ipcs_msg.dat* corresponds to a message queue.

MSGMNB

Maximum size (in bytes) of all the messages in a message queue. The column `CBYTES` reports the current size of a message queue.

MSGTQL

Maximum number of messages in the system. The total reported in the column `QNUM` for a particular measurement period corresponds to the total number of messages in the system.

MSGSSZ

Segment size in bytes.

MSGSEG

Maximum number of message segments in the system. The value of `MSGSSZ` multiplied by the value of `MSGSEG` gives the total memory space (in bytes) which is available for message data. The total value of the `CBYTES` column for a measurement period specifies the current total size of all the messages in the system.

5.6.3 Analyzing semaphores

You can also use the `sar` option `-m` to analyze semaphore activity. You may also evaluate the file `$HOME/measure/sar_bin.dat` which is generated by the procedure `main_meas` (see page 74):

```
# sar -m -f $HOME/measure/sar_bin.dat
```

Example output: see page 128

evaluation using `ipcs`

If the measuring procedure periodically starts the `ipcs -sa` command (as in the example procedure `ps_meas` on page 71), the file `$HOME/measure/ipcs_sem.dat` contains information about the semaphores. The information tells you which and how many semaphores exist and what their status is.

Example output:

IPC status from /dev/kmem as of Mon Jan 04 12:00:55 1999										
T	ID	KEY	MODE	OWNER	GROUP	CREATOR	..	NSEMS	OTIME	.
Semaphores:										
IPC status from /dev/kmem as of Mon Jan 04 12:01:27 1999										
T	ID	KEY	MODE	OWNER	GROUP	CREATOR	..	NSEMS	OTIME	.
Semaphores:										
IPC status from /dev/kmem as of Mon Jan 04 12:01:58 1999										
T	ID	KEY	MODE	OWNER	GROUP	CREATOR	..	NSEMS	OTIME	.
Semaphores:										
s	504	0x00004e20	--ra-ra-ra-	c1ladm	sapsys	c1ladm		20	11:02:21	...
s	505	0x00f42401	--ra-ra-ra-	c1ladm	sapsys	c1ladm		10	11:02:21	...
s	506	0x00004e82	--ra-ra-ra-	c1ladm	sapsys	c1ladm		1	no-entry	...
s	507	0x00007592	--ra-ra-ra-	c1ladm	sapsys	c1ladm		1	no-entry	...
s	508	0x00004e21	--ra-ra-ra-	c1ladm	sapsys	c1ladm		10	11:02:21	...
								.	.	.
								.	.	.
Message Queues:										

The following columns are of significance for evaluation:

NSEMS

The number of semaphores in the associated semaphore group.

OTIME

Time at which the last semaphore operation was applied to the associated semaphore group.

This evaluation enables you you check whether the configuration of the semaphores needs to be changed in the kernel parameters. When using semaphores you should ensure that you check the following kernel parameters against the measured values (see also page 194):

SEMMNI

Maximum number of semaphore groups

Each line of each measurement period logged in the file *ipcs_sem.dat* corresponds to a semaphore group.

SEMMNS

Maximum number of semaphores in the system

The total reported in the column NSEMS for a particular measurement period corresponds to the total number of semaphores currently present in the system.

SEMMSL

Maximum number of semaphores per group.

5.6.4 Analyzing shared memory

If the measuring procedure periodically starts the *ipcs -ma* command (as in the example procedure *ps_meas* on page 71), the file *\$HOME/measure/ipcs_shm.dat* contains information about shared memory. This information tells you which and how many processes work with shared memory.

Example output:

IPC status from /dev/kmem as of Mon Jan 04 12:00:55 1999									
T	ID	KEY	MODE	OWNER		NATTCH	SEGSZ	CPID	LPID .
Shared Memory:									
IPC status from /dev/kmem as of Mon Jan 04 12:01:26 1999									
T	ID	KEY	MODE	OWNER		NATTCH	SEGSZ	CPID	LPID .
Shared Memory:									
IPC status from /dev/kmem as of Mon Jan 04 12:01:57 1999									
T	ID	KEY	MODE	OWNER		NATTCH	SEGSZ	CPID	LPID .
Shared Memory:									
m	515	0x00000000	-Crw-rw-rw-	c1ladm		1	1	15339	15354 .
m	516	0x0382be84	-Crw-rw-rw-	c1ladm		15	1024	5352	15350 .
m	517	0x00000000	-Crw-rw-rw-	c1ladm		12	520	15353	15377 .
m	518	0x00000000	-Crw-rw-rw-	c1ladm		12	50204	15353	15377 .
m	519	0x00000000	-Crw-rw-rw-	c1ladm		12	1549800	15353	15377 .
m	520	0x00000000	-Crw-rw-rw-	c1ladm		2	662646	15354	15357 .
m	521	0x00000000	-Crw-rw-rw-	c1ladm		10	4000000	15360	15364 .
m	522	0x00000000	-Crw-rw-rw-	c1ladm		10	5000000	15360	15364 .
m	523	0x00000000	-Crw-rw-rw-	c1ladm		10	3000000	15360	15364 .
m	524	0x00000000	-Crw-rw-rw-	c1ladm		10	5120000	15360	15364 .
m	525	0x00000000	-Crw-rw-rw-	c1ladm		10	5000000	15360	15364 .
m	526	0x00000000	-Crw-rw-rw-	c1ladm		10	8192000	15360	15364 .
m	527	0x00000000	-Crw-rw-rw-	c1ladm		10	8192000	15360	15364 .
m	528	0x00000000	-Crw-rw-rw-	c1ladm		10	65540	15360	15364 .
m	529	0x00000000	-Crw-rw-rw-	c1ladm		10	98308	15360	15364 .
						.	.	.	.
Shared Memory:									

The following columns are of significance for evaluation:

NATTCH

The number of processes allocated to the shared memory segment in question

SEGSZ

Size of the shared memory segment

The ID field identifies the corresponding segment.

LPID

Process ID (PID) of the last process to use a shared memory segment

This evaluation enables you check whether the configuration of shared memory needs to be changed in the kernel parameters. When using shared memory you should ensure that you check the following kernel parameters against the measured values (see also page 195):

SHMMNI

Maximum number of shared memory segments

Each line of each measurement period logged in the file *ipcs_shm.dat* corresponds to a shared memory segment.

SHMMAX

Maximum size (in bytes) of a shared memory segment

The column *SEGSZ* specifies the current size of a shared memory segment.

SHMSEG

Maximum number of shared memory segments which can be simultaneously used by a process

6 Implementing tuning measures

This chapter describes the general measures which you may take to optimize the use of your system's main memory or of its hard disks or file systems. It also describes how you can enhance the performance of your computer by minimizing the number of running processes.

6.1 Processors

If you have identified CPU bottlenecks, the most effective solution is to install additional processors in your computer. However, this is only possible if your computer is able to accept further extensions.

You can also reduce processor load by reducing the number of processes (see section “Minimizing the number of processes” on page 151) or by taking care in the way you start processes (see below).

6.1.1 Binding processes to a CPU

An analysis of the measured results has revealed that:

- A process is using an exceptional amount of CPU time (see section “Identifying CPU-intensive processes” on page 88),
- no error (infinite loop or similar) has occurred.
- Then, if your computer is equipped with multiple processors (possible in following computers: RM200, RM300, RM400 as of model 730, RM600), you can bind the process in question exclusively to one CPU. The reserved CPU then works exclusively with the process which has been assigned to it. To do this, use the *mpcntl(1M)* command together with the option *-x*:

`mpcntl -x -ccpu [-ccpu ...] {-ppid [-ppid ...] | filename [options] }`



If you assign a process exclusively to one processor then the load borne by the other processors will be increased. Make sure that these processors are not overloaded following this operation.

- If you subsequently want to cancel this exclusive process-to-processor binding, you can use the *mpcntl(1M)* command with the option *-u*:

`mpcntl -u {-ppid [-ppid ...] | filename [options] }`

Example

During your analysis of the measured results, you have ascertained that the processes with the numbers 2222 and 4712 are using too much CPU time and that this is not due to the occurrence of an error. You therefore want to bind these processes exclusively to CPU number 1:

```
# mpcntl -x -c1 -p2222 -p4712
```

You can use the *mpcntl* command to check the CPU assignment:

```
# mpcntl -I
```

In the above example you would then see the following output:

CPU	...	#XBND
0	...	0
1	...	2

The column #XBND specifies the number of processes which are exclusively bound to the CPU named in the first column. Two processes are therefore exclusively bound to the CPU with the logical number 1.

Another way of starting suitably modified processes is provided by the *launchit(1M)* command. You can use this command to start a process which has been assigned certain characteristics, e.g. a defined priority.

6.1.2 Process scheduling

Another way of improving the performance of running processes lies in the manipulation of the process scheduler. This alternative may even be more effective than binding a process exclusively to one CPU.

For example, the process scheduler allows you to define real-time processes. Real-time processes have a defined response time within the system. By allocating different priorities you can achieve control of the process sequence and, in this way, influence the way in which processes run.

The commands *dispadm(1M)*, *nice(1)* and *priocntl(1)* are available for process control. You can use the *priocntl* command to change the priority of a command, even during runtime. If you use the *nice* command you can only set the priority at the start of a program.



Take care when manipulating process control. Oversights and incorrect operation may have a negative effect, and not only on the performance of your system.

Process control with `dispadmin`

The `dispadmin(1M)` command modifies your system's dispatcher tables. Any changes take immediate effect in the running system.

Example:

You want to display and modify the scheduling parameters using the `dispadmin` command. First of all you should display the configured scheduler classes:

```
# dispadmin -l
```

Example output:

```
CONFIGURED CLASSES
=====
SYS      (System Class)
FP       (Fixed Priority)
RT       (Real Time)
TS       (Time Sharing)
```

You can now use the `dispadmin -g -c class` command to output the scheduler parameters for the displayed classes, for example the real-time parameters which have been set:

```
# dispadmin -g -c RT
```

Example output:

```
# Real Time Dispatcher Configuration
RES=1000

# TIME QUANTUM          PRIORITY
# (rt_quantum)          LEVEL
1000                    #      0
1000                    #      1
1000                    #      2
...
800                     #     10
800                     #     11
...
100                     #     50
100                     #     51
100                     #     52
100                     #     53
100                     #     54
100                     #     55
100                     #     56
```

You can modify these parameters, for example using the `dispadmin -c RT -s rt.config` command. In this case, the current real-time parameters are overwritten with the values from the file `rt.config`.

6.2 Main memory

In Reliant UNIX 5.45, the main memory is dynamically managed. Consequently, there are few ways of influencing main memory operation. Despite this, you should ensure that utilization of the available main memory is optimized.

6.2.1 Storing data for longer in main memory

All accesses to a file system, pass via the system buffer. *BDFLUSHR* specifies the number of seconds after which the *fsflush* process is called in order to write a section of the modified data buffer in the file systems on the hard disk. The default values of *NAUTOUP* and *BDFLUSHR* (60 and 2 seconds respectively) mean that each time the *fsflush* process is called, 2/60 or 1/30 of the data buffer is checked. This does not represent a complete update of the file systems, but is used to prevent peak I/O load during *sync*.

NAUTOUP specifies the maximum number of seconds after which the file systems on the hard disk should be fully updated with information from the operating system data buffers (including the “old” buffer cache and inode cache). This ensures the consistency of the data on the hard disk (cf. *sync(2)*). For further information on the *BDFLUSHR* and *NAUTOUP* kernel parameters, refer to the chapter “Configuring the kernel” on page 185.

If you increase *NAUTOUP*, then the data is stored in main memory for longer. This has the advantage that the system needs to access the disk less often. However, such a modification also has its disadvantages:

- Disk activity is only deferred, not avoided, when the primary activity is the referencing of different data. In such cases, a considerably greater data volume must naturally be written to disk at *sync* times.
- If the system crashes then the data which is still in the buffer is lost.

6.2.2 Extending main memory

If you have detected that the system is paging out main memory pages or swapping out entire processes and that the kernel buffer is also running at full load, then the system is suffering from a memory bottleneck. The simplest, fastest and sometimes most economical way of overcoming this type of bottleneck is to install additional memory modules in your computer. This is only possible if there is still space for additional memory in your computer.

6.2.3 Reducing reserved shared memory

Software products exist which, by default, occupy a certain amount of main memory even if they do not need it. As this concerns virtual memory only, the main memory load is not increased. That of the swap device, however, is. For this reason, you should attempt to reduce the unneeded shared memory area. Check whether the software product provides special parameters or commands which can be used to reduce the reserved shared memory area. This is possible, for example, with INFORMIX-ONLINE and ORACLE.

It is particularly important to do this for program packets which are only rarely used.

6.2.4 Increasing the swap area

There are two cases in which you should increase the swap area:

- When you have detected the presence of a memory bottleneck following an analysis of the level of paging and swapping activity (see sections “Paging activity” on page 100 and “Swapping activity” on page 104).
- An analysis of the disk load reveals that the partitions which contain the swap area are heavily loaded (see section “Determining the disk load” on page 116).

In both cases you should check whether switching the swap area, or sections of it, to less busy hard disks results in an improvement in performance.

When doing this, observe the following rules:

- If possible, the size of the entire swap area should be equal to the expected memory requirement of all the processes. At the very least, the swap area must be as large as the physical main memory.
- The swap area should consist of a number of sections which are distributed over various disks and controllers.
- If possible, do not simply use one file within a file system as the swap area.
- If possible, do not locate a swap area on a disk which contains a frequently used file system or database area.
- If possible, do not locate a swap area on a disk containing the shared-access directories for executable programs (*bin* directories).



A system should always have enough main memory to prevent paging and swapping from causing a memory bottleneck.

6.3 Hard disks and file systems

Below we describe a number of measures which you can take in order to improve performance. Before you implement these measures, you should first make sure that the primary problem does not reside in a memory bottleneck (see page 136).

All the available hard disks should be subjected to uniform operation. If possible, programs, files and databases which are frequently accessed should be distributed across the various disks and file systems. Indeed, when you set up file systems you should take account of the number, size and access frequency of the files they contain. The actual configuration will depend on the application in question.

You should observe the following when distributing the hard disks to the SCSI chains and controllers: modern SCSI controllers can cope with the maximum number of connections per SCSI chain without performance being impaired. Performance considerations do not therefore necessarily dictate that the load should be distributed across multiple SCSI chains.

6.3.1 Selecting virtual disk management

The protection against failure, availability and performance of mass storage media can be improved through the implementation of RAID concepts (RAID = Redundant Array of Independent Disks). The Reliant UNIX operating system implements this concept by supporting virtual disks.

Virtual disks are formed by combining partitions (slices) from a number of physical disks to form a logical partition. The memory disk is a special type of virtual disk and may be used either fully or partly as a virtual disk. Read/write accesses to a virtual disk are significantly faster than in the case of a hard disk, with the result that processing speed is accelerated. Virtual disks are fully transparent for both users and applications.

Types of virtual disk supported

RM computers support five different types of virtual disk:

simple

Simple virtual disk

This type has the advantage that the disk can be subdivided into even smaller partitions than is possible with the physical disk.

concatenated

Concatenated virtual disk

This type is especially well suited to the processing of large data volumes. It combines a number of different partitions to form a logical storage area which may exceed the capacity of a physical disk and may even comprise more than 4 Gbytes.

striped

Striped virtual disk (RAID0)

In this type of virtual disk, the partitions are subdivided into stripes which are distributed across various disk drives and/or controllers (clusters). This can result in considerably improved throughput. If a large number of processes perform random access operations, the use of this type of disk may result in improved throughput

mirror

Virtual mirror disk (RAID1)

This type of virtual disk is used to improve the security of the available data. In addition, it improves performance when data is primarily accessed for reading. This type of disk mirrors two or more identical disk areas. Read accesses are always performed using the section with the lowest access time. Write accesses are performed almost instantaneously via a single system call.

Here it is particularly interesting to note that a memory disk can also be used as a mirror disk. In this way you can ensure that all read operations access main memory since this exhibits the shortest response time.

If either the original or the mirror disk fails, then the system “switches” to the other disk. This is done automatically when the system detects the error (see also the manual “Virtual Disks”). When the disk is again operational and activated, the system performs a catch-up from the current disk to the reactivated disk so that both disks, original and mirror are again identical.

statesave

Virtual disk with status information

When you use mirror disks you must also set up at least one *statesave* disk which contains information about the status of the mirror disk. It is, of course, advisable to mirror the *statesave* disk.

The *striped* disk type is the most interesting as far as tuning operations are concerned since it allows I/O jobs to be distributed across multiple disks.

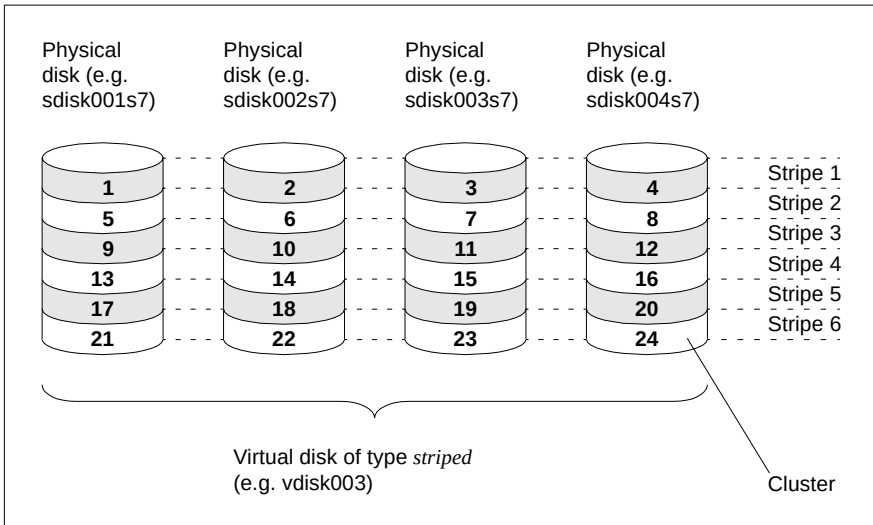


Figure 10: Structure of a virtual disk of disk type striped

Virtual disk configurations are described in the file `/etc/dktab`.

For example:

```
#ident "@(#) $Header: dktab,v 1.3 1993/02/08 $ SNI"
#
# Virtual disk configuration file
#
# The following are sample dktab entries.
# See dktab(4) for more information.
## A concatenated virtual partition.
/dev/vd/vdisk0 concat 3
    /dev/ios0/sdisk002s0
    /dev/ios0/sdisk002s1
    /dev/ios0/sdisk002s4
## A striped virtual partition with a cluster size of 32 blocks.
/dev/vd/vdisk1 stripe 2 32
    /dev/ios0/sdisk010s0
    /dev/ios0/sdisk010s1
## A statesave disk with two pieces.
/dev/vd/vdisk2 statsv 2
    /dev/ios0/sdisk002s15
    /dev/ios0/sdisk012s15
## A mirror disk with two pieces.
```

/dev/vd/vdisk3 mirror	2		
/dev/ios0/sdisk002s2	32	190000	
/dev/ios0/sdisk012s2	32	190000	

The meanings of the individual fields are as follows:

/dev/vd/vdisk0	Name of the virtual disk
concat 3	Virtual disk 0 is a concatenated disk consisting of three consecutive disk areas.
sdisk002s0	Names of the physical partitions which are to be combined to form a logical partition
sdisk002s1	
sdisk002s4	
/dev/vd/vdisk1	Name of the virtual disk
stripe 2 32	Virtual disk 1 is a striped disk consisting of two consecutive disk areas. The cluster size is 32 blocks each of 512 bytes.
sdisk010s0	Names of the physical partitions which are to be combined to form a logical partition
sdisk010s1	
/dev/vd/vdisk2	Name of the virtual disk
statsv 2	Status partition of the mirror disk /dev/vd/vdisk2This is required in order to store the status of the mirror disk(s). A <i>statesave</i> disk can store status information for multiple mirror disks. If multiple pieces are specified (as in the example), the status information is deposited on multiple disks.
sdisk002s15	Disk area in which the <i>statesave</i> disk is configured
sdisk012s15	This disk starts with an offset of 0. A <i>statesave</i> disk must be configured before the mirror disk. A single <i>statesave</i> disk can be used to manage multiple mirror disks.
/dev/vd/vdisk3	Name of the virtual mirror disk
mirror 2	Mirror disk with simple mirroring, i. e. an original area and a mirror of this area (copy).
sdisk002s2 0	Partitions containing the mirror disk and offset of the mirror disk (normally 0)
sdisk012s2 0	
190000	Size in blocks of the area to be mirrored
	The mirror area must be of exactly the same size as the area to be mirrored.

6.3.2 UFS and VxFS file system types

All connected file systems are listed in the file `/etc/vfstab`. The following types of file system are used:

- UFS UNIX File System; Default type for the root file system
- VxFS VERITAS File System; Default type for all other file systems



For information on VERITAS® file systems, refer to the chapter “Performance and tuning” in the manual “VERITAS File System (VxFS)”.

UFS type file systems optimize the position of data blocks during write operations: if possible, they attempt to store all the data blocks of a file in a single cylinder group. The size of the data blocks may be 4, 8 or 16 Kbytes as required. By default, RM computers use 16-Kbyte blocks. It is not possible to use larger blocks. A smaller block size may be advisable if the file system contains a large number of small files (e. g. scripts).

When a file system is generated, 10% of the space is reserved by default for optimization purposes. “Normal” users may only fill the partition containing their file system up to the value of 90% (of the space remaining available after the deduction of 10%). A higher filling level results in performance problems.

6.3.3 Minimizing program load operations

User programs are searched for and started in accordance with the specification of the `$PATH` environment variable. You should note the following points concerning the `$PATH` variable:

- No directory should be present more than once.
- The directories containing the most frequently used commands and programs should be entered first.
- The variable should contain only the most important directories.



Many applications demand that the directories in the path specification follow a particular sequence. You should therefore document any changes as otherwise you run the risk that some programs may no longer be locatable!

Example

For test purposes, enter the following command in a user's *.profile* file:

```
# echo "Path is: $PATH"
```

Evaluate the output.

Action

Set the *\$PATH* variable in accordance with the criteria described above.

6.3.4 Identifying users who occupy considerable disk space

If multiple users work with the system, you should regularly check which user is currently utilizing the most disk space. This is especially important if the fill level of a file system has almost reached the 90% limit or if it has already overflowed. In such cases you must reduce the level of disk utilization as quickly as possible.

You can use the *du(1)* command to determine the amount of disk space which is utilized. This command outputs the utilized disk space in blocks of 512 bytes. You can also enter the command in a small shell script as the following example demonstrates (in this case for the */home* file system):

```
# cd /home
# for i in *
> do
> du -s $i
> done | sort -rn
```

Example output:

```
329996 kfischer
69360  mar
55140  gT$IMX
15980  mar2
13708  TPCA-INF
4240   nerv
3672   vmsys
3196   TMPDIR
3184   mm
376    tools
108    oasys
40     xuser
32     xmeter
32     lost+found
32     admin
28     krenz
12     isc
```

8	spec
---	------

```
8      hk
8      ate
4      informix
0      rm
0      rb
```

This example clearly shows that in this case the *kfisher* user is using the most disk space. The system administrator should contact this user to discuss whether some of the files can be transferred elsewhere, deleted or compressed.

6.3.5 Searching for large files

Large files are also easy to find. To do this you can use the *find(1)* command. It may even be advisable to extend the search across the entire system since *log* and *error* files can grow to an astonishing size. The following example lists all the files which contain more than 2000 blocks of 512 bytes (approximately 1 Mbyte):

```
# find . -size +2000 -print | xargs ls -l
```

It may be advisable to compress large files which are not frequently accessed. In the case of data files, this can result in a saving of up to 70% of the utilized disk space. The disadvantage is that you will need to uncompress these compressed files before working with them again.

You use the commands *compress(1)*, *uncompress(1)*, *pack(1)*, *unpack(1)*, *pcat(1)* and *zcat(1)* to perform these operations. The syntax of the *compress* and *uncompress* commands is presented below by way of example:

compress -v file

uncompress file.Z

file is the name of the file which is to be compressed. *compress* then generates a compressed file which has the name extension *.Z*.

6.3.6 Searching for old files

Access to UFS type file systems is considerably faster when the maximum possible amount of free storage space is available. You can check this value using the *df* command. At the very latest, when the fill level reaches 80%, you should attempt to remove old files and software packages which are no longer needed. As of a fill level of 85% significant performance problems may arise.

Example

Identify the user files in */home/xy* which have not been modified for the last 200 days and which have not been accessed for the same period. You can use the *find(1)* command to perform this task:

```
# find /home/xy -mtime +200 -atime +200 -print
```

The primary expressions *-mtime* (*m* = modification) and *-atime* (*a* = access) allow you to output all those files which possess a particular modification and access date. The specifications have the following syntax:

-mtime ±n or ***-atime ±n***

-mtime applies if the file in question was last changed *n* days ago (+ more than, - less than). *-atime* applies if the file in question was last accessed *n* days ago (+ more than, - less than).

Action

Consult the users and save these files to tape or diskette before deleting them.

6.3.7 Relocating user data to free file systems

If one file system is overused while there is still ample free space in other file systems, such as the */home1* area, you may relocate an entire user area.

For example: Move the user area */home/xy* to */home1/xy*.

```
# cd /home
# find xy -print -depth | cpio -pdm /home1
# rm -rf /home/xy
```

This moves the user area *xy* from the */home* file system to the */home1* file system. However, you must now use the *usermod* command to change the login directory. If programs etc. contain the absolute pathname */home/xy* you must change every occurrence of this to */home1/xy*.

6.3.8 Reducing the time taken to search directories

The file system implementation checks every element in a path from left to right and searches for each element in the directory hierarchy. If the element is a symbolic link, then the path of the symbolic link is checked element by element. In order to check a single path element it may be necessary to read multiple directory blocks from the hard disk if the element is not found in the DNLC (see below). Consequently, long pathnames and pathnames which contain symbolic links may be relatively time-consuming for the system as a whole.

Paths which are searched very frequently should not contain symbolic links. The same applies, for example, to directories which contain general commands and to temporary directories.

If it is possible to generate a symbolic link which uses a relative path instead of a longer, absolute path then you should favor the relative path. In the example below, variant 1 should be preferred to variant 2.

Variant 1: /usr/src/nopkg/lib/filter.c
 -> ../bin/filter.c

Variant 2: /usr/src/nopkg/lib/filter.c
 -> /usr/src/nopkg/bin/filter.c

If you are subdividing an existing file system you should again prefer variant 1 to variant 2.

Variant 1: move the directory hierarchy to a new file system and mount the new file system at the old position,

Variant 2: move the directory hierarchy to a different file system and generate a symbolic link from the old to the new position.

You can use the *sar -a* command to find out whether the time required to search directories is excessive (see section “Determining file access frequencies” on page 122).

6.3.9 Length of file and directory names

When files and directories are accessed, the system first looks for the name in the Directory Name Lookup Cache (DNLC) where they can be found more quickly since path names do not have to be resolved. If a name is not found in the DNLC it is entered in it. If necessary, the least used entry is replaced by the new entry.

The maximum number of entries in the DNLC is specified in the *NCSIZE* kernel parameter (see page 187). However, the DNLC can only store names which consist of no more than 25 characters. This maximum length is fixed in the *NC_NAMLEN* kernel parameter (see page 186). Names which are longer than *NC_NAMLEN* have to be resolved each time they are accessed and consequently require extra processing time.

6.3.10 Modifying optimization strategies for UFS file systems

There are two ways in which a file system can be organized:

- so that it occupies as little storage space as possible, or
- so that directory access times are minimized.

You should always plan your strategy to achieve optimum access times.

Analysis

The *mkfs(1M)* command with the option *-m* outputs the settings used to generate a file system.

Example: Disk partition */dev/ios0/sdisk011s5* (i.e. */home*)

```
# mkfs -F ufs -m /dev/ios0/sdisk011s5
```

Example output:

```
mkfs -F ufs -o nsect=26,ntrack=15,bsize=16384,fragsize=2048,cgsize=16,free=10,rp
s=59,nbpi=2048,opt=s,apc=0,gap=8 /dev/ios0/rsdisk000s5 377520
```

The following outputs are of importance for the analysis of the optimization strategy:

```
free
    Reservation for optimization

opt
    Optimization strategy:
    s      reduce storage space to a minimum
    t      minimize access times
```

Action

Use the `tunefs(1M)` command to set the optimization strategy **opt=t** (minimized access time). The file system must be unmounted before you can change the strategy.

Example: `/home` is located under `/dev/ios0/sdisk000s5`

```
# umount /home
# tunefs -o time /dev/ios0/sdisk000s5
# mount /home
```

6.3.11 Determining the block size of a file system

If an application is reading very large files, read access can be made faster by specifying as large a block size as possible when setting up a UFS file system. This means that larger volumes of data can be read simultaneously.

The effect that the block size of a VxFS file system has on I/O performance is described in the manual “VERITAS FILE System (VxFS)”.

You can set the block size of a file system with the `df -g` command.

6.3.12 Installing additional hard disks

If a very large number of disk accesses are performed and if utilization of available hard disk capacity is approaching 70%, the use of further disks considerably reduces hard disk load.

6.3.13 Balancing hard disk utilization

User software and data should be optimally distributed across all the disks so that all operations are not concentrated on a single disk. If possible, data areas which are frequently accessed should be distributed across as many physical disks as possible. Another possibility is to use *striped* virtual disks. For more information refer to the section “Selecting virtual disk management” on page 138.

6.4 System tables

It is not normally necessary to modify the kernel parameters for the system tables. However, should it prove necessary to do this, refer to the chapter “Configuring the kernel” on page 163 onwards for information on the procedure.

In Reliant UNIX, some system tables and kernel parameters are dynamically enlarged or reduced (e.g. the process table). Dynamic tables always occupy exactly as much storage space as they require. Such tables do not therefore need to be optimized. Dynamic tables will not overflow until the maximum value has been reached. However, there are also static kernel parameters whose values it may be advisable to increase.

Modifying i-lists (inode tables)

The table of active inodes is one of the static kernel parameters. The kernel parameter concerned is *UFSNINODE*. If the table limit for UFS type file systems has nearly been reached, increasing this value can result in a significant improvement in performance! To do this, proceed as follows:

- Increase the value of *UFSNINODE* to three times the current value using the *idtune(1M)* command. To ascertain the current value of the parameter, inspect the file */etc/conf/cf.d/mtune* or */etc/conf/cf.d/stune* (see section “Determining the values of kernel parameters” on page 168).
- Next, you must regenerate the kernel (see chapter “Configuring the kernel” on page 163) and reboot the system.
- Finally, perform a new load measurement and test the application.

6.5 Minimizing the number of processes

Only processes which are required for the application should be active. You should therefore deactivate all the active processes relating to unneeded software together with all other superfluous processes (e. g. *cron* processes). You should also check whether it would be possible to run memory-intensive programs during periods of low system load, e. g. at night or over the weekend.

6.5.1 Deactivating unnecessary timing jobs

Check closely which timing jobs could be omitted in the directories */var/spool/cron/atjobs* and */var/spool/cron/crontabs*. For example, you can completely deactivate the default system performance monitoring which is evaluated by the *sar* command. To do so, turn the lines for periodically started *sar* processes in the file */var/spool/cron/crontabs/sys* into comment lines. To do this enter a “#” at the start of the relevant lines:

```
# * * * 0-6 /usr/lib/sa/sa1
# 20,40 8-17 * * 1-5 /usr/lib/sa/sa1
# 5 18 * * 1-5 /usr/lib/sa/sa2 -s 8:00 -e 18:01 -i 1200 -A
```

The modifications do not take effect in the individual “crontabs” until the *cron* process is restarted. Before this can be done, it must first be stopped:

```
# sh /etc/init.d/cron stop
# sh /etc/init.d/cron start
```



Do not expect to gain too much from deactivating the performance monitoring. This uses an interval of 20 mins. Instead of this, you should search for further *cron* jobs which can be omitted.

6.5.2 Deactivating uucp

If you do not need the *uucp* software you can deactivate it (UUCP = UNIX to UNIX Copy; connection between two UNIX computers). To do so, turn the lines for periodically started *uucp* processes in the file */var/spool/cron/crontabs/root* into comment lines. To do this enter a “#” at the start of the relevant lines:

```
# 48 11,14 * * 1-5 /bin/su uucp -c "/usr/lib/uucp/uudemon.admin > /dev/null 2>&1"
# 45 23 * * * ulimit 5000; /bin/su uucp -c "/usr/lib/uucp/uudemon.cleanup > /dev"
# 40 * * * * /bin/su uucp -c "/usr/lib/uucp/uudemon.poll > /dev/null"
# 26,56 * * * * /bin/su uucp -c "/usr/lib/uucp/uudemon.hour > /dev/null"
```

The modifications do not take effect in the individual “crontabs” until the *cron* process is restarted (see above).

6.5.3 Deactivating unrequired daemon processes

You can use *ps -ef* to determine the processes which are active or present throughout the system. By deinstalling entire software packages, you also remove the corresponding activation routines from the */etc/rc* directories.

6.5.4 Deinstalling software which is no longer required

Produce an overview of the installed software. To do this, you can use the *sysadm*(1M) operating environment (menu items *software – list*). Depending on the configuration, you may be able to deinstall software packages which are no longer required (menu items *software – remove*). Instead of the operating environment, you may also use the *pkginfo*(1M) and *pkgrm*(1M) commands for the same purposes.

If you deinstall software packages which are linked to the operating system, do not forget to generate a new kernel and to restart the system (see section 7.5 on page 177).

6.6 Optimizing compilers

Compilers have a significant impact on the performance of application programs. With current processors, machine-dependent optimizations are more important than general, machine-independent optimizations. You should therefore use the compiler that produces the best possible code for the destination machine by taking account of machine-specific features:

- Hardware features
- CPU-specific heuristics

The compiler should also provide the latest optimization procedures (e.g. feedback optimization, see below). It is particularly important when it comes to demanding applications (databases, complex process controllers) that the compilation process be thought through properly.

Feedback-based optimization

Feedback-based optimization uses statistical data, which is recorded in a special program run with the instrumented unoptimized program, to provide more information about the typical behavior of the program and to tailor optimization to this information.

The following optimizations can be performed using statistical data:

- Feedback-controlled inline generation
- Function positioning
- Branch elimination
- Branch prediction
- Function splitting

This technology allows the program to be specially adapted to the future application area. Large applications can also be optimized in this way, although the compiler time increases by a correspondingly large volume (more than 4 hours for a database, 10 Mbytes binary).

Detailed compiler documentation is available which provides information about using feedback.



The compilers offered by Siemens (C-DS 1.2, C-DS++, pyrC5.0, pyrC6.01) support feedback-based optimization.

6.7 TCP/IP and NFS tuning

Both of these topics are too complex to describe in detail in this manual. Nevertheless, we will provide some important though concise information below with regard to tuning. For further details, please refer to the “Network Administration” manual and the “Network Programming Interfaces” manual.

6.7.1 The enhanced protocol stack

In Reliant UNIX versions before Version 5.45, the TCP/IP protocol stack was available in two variants. The first was based on 4.3 BSD and the other was based on Streams. The Streams implementation was originally a component of UNIX SVR4 (System V Release 4). This meant that there were many TCP/IP modules available in both variants. In comparison with the Streams stack, the

BSD stack mainly offered better performance for the socket interface and, in exceptional cases, was better adapted to the original semantics of the socket interface, which was originally designed for a non-Streams environment.

In Reliant UNIX 5.45, the TCP/IP protocol stack was redesigned (TCP/IP modules are based on 4.4 BSD and these modules occur only once in the system). This redesigned stack is referred to here as the enhanced protocol stack. The Streams interfaces for TPI, TLI and XTI are made available via adaption modules at the beginning of the TCP/IP layer.

The enhanced protocol stack was carefully designed to ensure compatibility with the applications available. For special cases in which the exact behavior of Reliant UNIX versions before Version 5.45 is required, it is possible to revert the system back to the old dual stack implementation. This is referred to here as the fallback solution. Reliant UNIX 5.45 is shipped with the enhanced stack as standard.

For details on the installation of an enhanced or fallback kernel, see the sections 3.4.3.5 and 3.4.3.6 of the “Network Administration” manual. The *test_enhanced* command has been introduced for the purposes of checking which kernel (enhanced or fallback) is running.

The enhanced stack has settings for running a socket application with the BSD or Streams variant. This was also possible in earlier system versions using the *LIBSOCKET_BSD* environment variable. However, Reliant UNIX 5.45 has more advanced setting options, both globally in the system and on an application-specific basis.

Applications which use sockets and which assume that the implementation of these sockets is STREAMS-based must be started with the *LIBSOCKET_BSD=N* environment variable in Reliant UNIX 5.45 with the enhanced kernel. If this does not happen, kernel sockets are made available by default, i.e. applications run without the Streams application modules and STREAMS-specific operations such as *I_POP*, *I_PUSH* or *I_LINK* cannot be executed.

There is another option, whereby the default mode of the sockets generated with *socket()* calls can be changed globally in the system:

- The configuration file */etc/inet/socklib.conf* describes the default behavior.
- The file */etc/socklib.conf.stream* describes a behavior that would have existed in 5.43/5.44 systems.

The file *socklib.conf* is described in Chapter 3.4. of the “Network Administration” manual, while BSD and STREAMS are described in Chapter 3.6 of the “Network Programming Interfaces” manual.

6.7.2 Performance of the enhanced stack

For socket applications with the BSD variant, the enhanced stack provides a level of performance which is equally as good as for the BSD fallback stack in Version 5.45 or the BSD stack in previous versions, while it provides considerably better performance for socket applications with the Streams variant and for TLI and XTI applications. TLI and XTI applications can take advantage of these performance enhancements without the need for any specific settings.

Depending on the setting (see the “Network Programming Interfaces” manual, Section 3.6.1.2) applications which previously ran via the Streams stack can run with the BSD variant and thus offer the best possible performance. Only when these applications run with the Streams variant, as a result of a changed global system setting or an explicit specification (by setting `LIBSOCKET_BSD=N`), are they less efficient than an application with a BSD variant, although they are still considerably faster than in the previous version.

Applications should run using the BSD variant wherever possible in order to achieve the best possible performance.

6.7.3 Effect of the socket options `SO_SNDBUF` and `SO_RCVBUF` and the routing metrics

For more information on this topic, refer to the “Network Programming Interfaces” manual, Chapter 3.4.8, Section “The socket options `SO_SNDBUF` and `SO_RCVBUF`”, as well as the “Network Administration” manual, Chapter 11.4, “Routing metrics”.

6.7.4 Commands for resolving performance bottlenecks

6.7.4.1 `netstat/enetstat`

The most important tool is the `netstat(1M)` command, while for special files it is the `enetstat(1M)` command.

For each LAN interface, `netstat -i` returns, among other things, the number of sent and received packets as well as the corresponding number of errors. The `etherstat(1M)` command is useful for obtaining detailed analysis of error sources.

`netstat -s` provides data concerning the various protocol layers, such as TCP, UDP and IP.

`enetstat -s` provides the same basic data as `netstat -s`; however, the output is coordinated with kernels using the enhanced protocol stack, which means that some new counters are displayed, while others are not.

Evaluation of the statistical output always presents a problem, as the counters that are output provide summary information concerning all available network interfaces since the last system start or since the deletion of this data with an appropriate command. To analyze concrete problems, it is therefore advisable to record the current status, reproduce the problem and to then record the subsequent status. The comparison of both statuses can then be used for diagnostics purposes.

Some counters are explained below. The counters are retained in the system as unsigned 32-bit volumes. Overflows can occur quite easily, particularly in the case of byte counters.

TCP

The mode of operation of a TCP endpoint is influenced not only by data transfer, but also by time constraints. These are known as “timers”. From the point of view of performance, the *Retransmission timer* is of particular interest. It defines when an attempt to re-send a packet is to be made in the case of a TCP connection. The work of the *Retransmission timer* is supported by the *Fast retransmit* mechanism which attempts to determine, as quickly as possible, whether data packets are to be sent again, thus minimizing the negative effects of packet losses on the throughput of a TCP connection.

embryonic connections dropped

Shows the number of attempts to address a non-supported service on another system from the local system.

retransmit timeouts

Number of times the packet was re-sent, because acknowledgement was not received from the partner for this packet within the period specified with the *Retransmission timer* (see also the *data packets retransmitted* counter).



In a normal situation, where there are no sporadic packet losses in the network, the increase in the number of *retransmission timeouts* is caused mainly by failed connection setups, i.e. by the attempt to set up a TCP connection to partner system that cannot be accessed.

connections dropped in rxmt timeout

Number of connections which had to be cleared down because of *retransmission timeouts*. This type of connection cleardown takes place if the attempt to send a packet for a connection has still not resulted in a positive acknowledgement from the partner after multiple retries.

total packets sent

Total of all TCP packets sent.

data packets sent

Total of all TCP packets sent with user data.

data bytes sent

Total volume of user data sent via TCP. The division with the *data packets sent* counter can give an idea of the average user data volume of a TCP packet.

data packets retransmitted

Number of data packets that are re-sent. The difference between this counter and the *retransmit timeouts* counter shows the frequency with which the *Fast retransmit* mechanism is active.

data bytes retransmitted

Total volume of user data for TCP packets sent as a result of retries. A high rate of re-sent packets indicates bottlenecks or problems in the local system, in the network, or in one or more partner systems.

persist timeouts

Number of *persist timeouts*. A TCP endpoint is in persist mode if it has data to send but the partner is not currently ready to receive additional data. This may be the case if, for instance, the corresponding application on the partner system is not responding. The number of *persist timeouts* corresponds to the number of sent window probes.

timeout in persist state (only with *enetstat -s*)

Number of connections that were cleared down as a result of *persist timeouts*. On Reliant UNIX systems, the continued lack of a response to a sent window probe is evaluated in the same way as the continued lack of an acknowledgement in the event of a retransmission, therefore leading to a connection cleardown. If a window probe from the partner is responded to in the sense that the partner acknowledges receipt of the packet, but does not indicate that it is ready to receive additional data, this is not interpreted as “no acknowledgement”, and does not therefore lead to a connection cleardown, except in the case of *persist timeouts on FIN_WAIT_1* (see next counter).

persist timeouts on `FIN_WAIT_1` (only with `enetstat -s`)

Number of connections that had to be cleared down in the *Linger-On-Close* state. This happens if, after a local `close()` on a socket with `SO_LINGER` activated, all data could not be sent to the remote partner after the linger timer expired and the partner responded to the sent window probe, but did not open its receive window. An endpoint in the `FIN_WAIT_1` state then exists on the local system, whereby this endpoint is no longer linked with an application. The data linked with this endpoint is discarded after the *persist timeout* expires and the endpoint is removed from the system.



For the local system, a high number of *persist timeouts* does not mean that there is a certain system load, but indicates problems relating to the applications, between which the data is to be transferred. The data flow between the applications is blocked, which means that the application receiving the data no longer processes it.

UDP

`netstat -s`: packets dropped because upstream queue was full

`enetstat -s`: packets dropped because input socket was full

Number of received datagrams which could not be forwarded to the application because of an overflow in the corresponding socket buffer.

This is caused by the application processing the received datagrams too slowly.

input packets (only with `enetstat -s`)

Total number of received UDP datagrams.

packets dropped because no socket on port

Number of datagrams that had to be discarded because they could not be assigned to any recipient.

bcast packets dropped because not socket on port

As above, except that the IP destination address specified in the datagram was a broadcast address.



If there is a negative ratio of discarded to received packets, the local system is loaded with the other users in the network without the packets actually containing any useful data.

IP

fragments received

Number of fragments received

fragments dropped (dup or out of space)

fragments dropped after timeout

Number of discarded fragments



If there is a negative ratio of discarded to received fragments, this indicates that the splitting up of IP datagrams into multiple fragments or their transfer in the network frequently leads to data (fragment) losses.

In many systems, the utilization of NFS is the most important reason for the generation of fragmented IP datagrams and may indicate the necessity for their reassembly. In such an environment, a high number of rejected IP fragments may be the reason behind problems associated with NFS throughput. Reducing the block size that NFS works with can result in an improvement, provided the rate of rejected IP fragments drops considerably after such action is taken.

total packets reassembled ok (only with *enetstat -s*)

Number of IP datagrams that could successfully be reassembled from multiple fragments. From the ratio to the number of successfully received fragments, it is possible to determine the average number of fragments into which an IP datagram was split.

input packets dropped because upstream queue was full

(only with *netstat -s* and the fallback kernel)

Number of IP datagrams that could no longer be sent upstream to the transport protocols because of an overflow on the local system.

packets not sent because downstream queue was full

(only with *netstat -s* and the fallback kernel)

Number of IP datagrams which had to be discarded in the send direction on the local system because the corresponding network interface was no longer able to receive data.

6.7.4.2 etherstat

Statistical data, which allows certain conclusions to be drawn in relation to the cause of packet losses, is provided for each interface.

```
> etherstat zx0
zx0 (08-00-06-0d-67-03):
    0 receiver overflows
    0 receiver CRC errors
    0 receiver alignment errors
    0 receiver queue full
  935262 receiver packets for unknown sap
    0 receiver packets discarded for other software reasons
 35997003 good packets received
    0 transmitter underflows
 4570882 transmitter collisions
    0 transmitter excessive (16) collisions
    0 transmitter packets discarded for other software reasons
 41658997 good packets transmitted
```

receiver overflows

Number of packets which could not be received because of an overflow of the local FIFO stack on the LAN chip.

This generally indicates DMA problems, i.e. data transfer to the main memory from the LAN controller is interrupted. These problems may be caused by a high parallel LAN or SCSI load, especially if this load occurs on the same HIOS or EHIOS board containing the relevant LAN controller.

receiver CRC errors

receiver alignment errors

Number of packets which could not be received correctly because of cyclic redundancy check errors.

This is generally caused by

- transmitter underflows on the sending machine
- poor cabling (defective, too long, damping too high)
- defective controllers
- problems with switches and routers on the path to this system

The same errors can be observed if the partners have different settings for the speed or the mode (half-duplex, full-duplex). The current settings can be checked with the *etherstat -v* command.

receiver queue full

Number of received packets which, for software-related reasons, could not be forwarded because of overflows in the queues between driver and IP.

receiver packets for unknown sap

Number of received packets which do not belong to the TCP/IP protocol family.

receiver packets discarded for other software reasons

Number of received packets which could not be processed.

Reasons for this may be:

- a memory bottleneck caused by a high load situation
- overflows in the queues above the driver
- LLC header too short

good packets received

Number of received packets.

transmitter underflows

Same as *receiver overflows*, but in the send direction, i.e. DMA is from the main memory to the LAN controller in this instance.

transmitter collisions

Number of collisions detected during the sending of a packet.

The effect of collisions in the case of 10/100 Mbit Ethernet is not too serious, as they generally only lead to a small delay in the send operation. However, *transmitter excessive (16) collisions* are more problematic, as they tend to lead to packet losses.

transmitter excessive (16) collisions

Number of packet losses in the send direction, 16 of which in succession result in a collision being detected.

The most common causes of a high collision rate are too large a network or defective controllers.

transmitter packets discarded for other software reasons

Number of sent packets which could not be processed.

Reasons for this may be:

- a memory bottleneck caused by a high load situation
- overflows in the queues between driver and IP
- no authorization for sending broadcast packets

good packets transmitted

Number of sent packets.

6.7.5 Slot for zx subcontrollers

receiver overflows (see previous section dealing with etherstat) lead to packet losses and LAN performance may suffer considerably if the error rate is correspondingly high ($> 1/10000$ in the case of 100 Mbit Ethernet).

For this reason, zx subcontrollers (EIFC-HC) with high performance requirements are best installed on HIOS or EHIOS boards which result in little or no SCSI load. High parallel data streams through LAN and SCSI, such as those used in the case of a Networker network backup, in which both data streams (receive via LAN + save via SCSI) run via a HIOS board, can cause a high rate of *receiver overflows* and thus have a detrimental effect on LAN performance.

You should always ensure that a fast tape device is not operated in parallel with a zx subcontroller with high performance requirements on the same HIOS or EHIOS board.

6.7.6 NFS and RPC

For information on the options available for optimizing NFS and RPC, refer to Chapter 6.8, "Optimizing operation of NFS" of the Network Administration manual for Reliant UNIX 5.45A0.

The following information is not contained in the above manual, as the corresponding functionality is only available with Reliant UNIX 5.45A10 or later: The data from NFS read and write jobs is transferred in blocks of up to 32 Kbytes with NFS V3 in the case of Reliant Unix 5.45A10 or later. The *rsz* and *wsz* options of the *mount(1M-nfs)* command allows block sizes smaller than the above maximum value to be set.

7 Configuring the kernel

The kernel manages internal tables in which it makes entries during operation. The size of these tables and lists as well as the upper limits of certain memory areas are defined by special kernel parameters, the tunable parameters. You can use these parameters to adapt the Reliant UNIX operating system to a certain extent to your particular operating conditions. This chapter describes how you modify the kernel parameters and generate a new kernel. This is followed by a description of the individual parameters.

7.1 The mtune file

All the tunable parameters are defined in the file *mtune*. This file is located in the configuration directory of the kernel */etc/conf/cf.d*. The parameters which are modified by the system administrator are located in the file *stune* which can be found in the same directory. The format of *mtune* is described under *mtune(4)* in the “System Administrator’s Reference Manual”. Each entry in *mtune* consists of four specifications: name of the parameter, default value, minimum value and maximum value. The default value may also be a formula. Below is an extract from *mtune*:

```
*SECTION 2: other relevant parameters for tuning
*=====
*
*The following parameters can be adjusted to individual configurations.
*This should be done when
*   - lack of an resource results in an error message,
*   - performance tuning is wanted.
*
*
* General Kernel Parameters -----
*
NCALL      (NPROC+16)      30      21861
* NPROC is limited by the amount of kernel virtual address space
* which is allocated for mapping the u areas (384 MB).
* Since each u area requires 32 KB, the maximum number of processes is
* limited to 384 MB / 32 KB = 12308
NPROC      (MAXUSERS*8+20) 100      12308
MAXUP      40      15      8192
NHBUF      1024     128     2048
NPBUF      200      20      2000
```

The kernel parameters are traditionally subdivided into the following parameter types:

- General kernel parameters
- File system parameters
- Paging parameters
- STREAMS parameters
- Message parameters
- Semaphore parameters
- Shared memory parameters
- Device driver parameters
- Timer and scheduler parameters
- Resource limit parameters
- Specific parameters in Reliant UNIX

Not all the parameters have the same significance for the system. Consequently, the *mtune* file contains a number of parameters which have a considerable influence on system performance. Other parameters may not be changed and yet others are not even used by the kernel.

In order to illustrate the varying significance of the kernel parameters for tuning, the file *mtune* is subdivided into eight separate sections:

SECTION 1	The most relevant parameters for tuning
SECTION 2	Other relevant parameters for tuning
SECTION 3	Seldom or never changed tuning parameters
SECTION 4	Machine-type specific parameters
SECTION 5	Parameters kept for compatibility reasons
SECTION 6	Parameters for special test purposes only
SECTION 7	Parameter definitions for special products
SECTION 8	More machine-type specific parameter values

Section 1 lists those parameters which are most frequently used for tuning purposes. This section differs from the other *mtune* sections in that the parameters or parameter types in question are not defined but simply named. The definitions themselves can be found in sections 2 and 8 of *mtune*.

The following parameters are important for the adaptation of tables and limit values to individual requirements as well as for general tuning measures:

Parameter	Parameter type	
<i>MAXUP</i>	General kernel parameters	186
<i>MAXUSERS</i>	"	205
<i>NPROC</i>	"	188
<i>NAUTOUP</i>	"	186
<i>BDFLUSHR</i>	"	185
<i>BUFHWM</i>	"	185
<i>NPTY</i>	"	202
<i>NPTEM</i>	"	202
<i>NINODE</i>	File system parameters	190
<i>MSG...</i>	Message parameters	193
<i>SEM...</i>	Semaphore parameters	194
<i>SHM...</i>	Shared memory parameters	195
<i>SFNOLIM</i>	Resource limit parameters	199

Table 7: The most important kernel parameters for tuning operations

7.2 Reasons for modifying the kernel parameters

The default values of the kernel parameters are perfectly satisfactory for the majority of applications. There are only two cases in which it may be advisable to modify them:

- The kernel is no longer operating efficiently because the “old” buffer cache is too small. This cache is always located in the *seg_kmem* segment (see figure 3 on page 10). It is used for file system management (inodes, indirect blocks and file system management blocks).
- An application’s system calls fail because resources which are specified by the kernel parameters are exhausted (e.g. semaphores or message queues).

Of particular importance is the configuring of interprocess communications (message queues, semaphores, shared memory) via kernel parameters. If an application system makes intensive use of interprocess communication mechanisms then it is essential that you examine the message queues, semaphores and shared memory load when analyzing the overall system.

Modifying the kernel parameters may have a significant impact on system performance, both positive and negative. You should therefore check which applications are running on your computer and analyze the computer's current performance. Before changing any kernel parameters you should take other performance factors into consideration, such as, for example, the organization of the data on the hard disks and the efficiency of the *\$PATH* specifications.

7.2.1 Special requirements

If your application possesses special performance requirements, you may need to experiment with various combinations of parameter values. The following table provides a summary of the kernel parameters which need to be changed to meet particular performance requirements.

Requirements placed on the operating system	Parameter to be changed	
It must be possible to generate large files.	<i>SFSZLIM</i> , <i>HFSZLIM</i>	199
All users should be able to open more files.	<i>SFNOLIM</i> , <i>HFNOLIM</i>	199
All users should be able to execute more processes.	<i>MAXUP</i>	186
More processes should be able to execute simultaneously throughout the system.	<i>MAXUSERS</i> , <i>NPROC</i>	205 188
The automatic <i>sync</i> rate should be changed.	<i>NAUTOUP</i> , <i>BDFLUSHR</i>	186 185

Table 8: Modifiable kernel parameters, part 1

For example, you may want to be able to create very large files. As Superuser you can achieve this by modifying the *ulimit* value for a particular shell process (e.g. using the shell command *ulimit(1)* or by modifying the *ULIMIT* entry in the file */etc/default/login*). An alternative solution is to modify the kernel parameter *SFSZLIM* for all users (see page 199).



Caution: The value of *SFSZLIM* is overwritten by the *ULIMIT* entries in the files */etc/default/login* and */etc/default/init*.

You can use the *pfenable(1M)* and *launchit(1M)* commands to permit users who do not possess *root* permission to run an unlimited number of processes.

7.2.2 Overwriting the operating system limit values

It may happen that certain operating system limit values are exceeded. In such a case, a message to this effect is output at the console. Some of these messages simply take the form of a comment while others precede a system crash which is accompanied by further diagnostic messages and requires you to restart the system. If you receive one of the messages which is listed in table 9, please refer to the description of the corresponding kernel parameter for further information.

Kernel message	Parameter to be changed	
Attempting to link in more shared libraries than system limit	<i>SHLBMAX</i>	189
dquot table full	<i>NDQUOT</i>	190
File too large	<i>SFSZLIM</i>	199
newproc - no procs	<i>NPROC</i>	188
No ptem structures	<i>NPTEM</i>	202
No such device	<i>NPTY</i>	202
No such device or address	<i>NLOG</i>	192
Not enough space	<i>SDATLIM,</i>	199
	<i>SVMMLIM</i>	200
Out of stream resources	<i>NAUTOPUSH,</i>	191
	<i>STRTHRESH</i>	193
swapdel - too few free pages	<i>MINASMEM</i>	191

Table 9: Modifiable kernel parameters, part 2

Kernel message	Parameter to be changed	
Timeout table overflow	<i>NCALL</i>	186
Too many open Files	<i>SFNOLIM</i>	199
ufs_iget - inode table overflow	<i>UFSNINODE</i>	190
v.v_phash adjusted to <i>number</i>	<i>NPROCHASH</i>	188

Table 9: Modifiable kernel parameters, part 2

7.3 Determining the values of kernel parameters

You can find out the value which is entered for a parameter in the files *mtune* and *stune* by using, for example, the *egrep(1)* command:

```
# egrep NPROC /etc/conf/cf.d/mtune /etc/conf/cf.d/stune
```

Example output:

```
mtune:* NPROC      - maximum number of processes that can run in parallel
mtune:NCALL        (NPROC+16)      30      21861
mtune:* NPROC is limited by the amount of kernel virtual address space
mtune:NPROC        (MAXUSERS*8+20) 100     12308
mtune:FLCKREC      (16*(NPROC+MAXUSERS+16)/10+72) 40      51433
mtune:NPROCHASH    64      2      1024
mtune:NINODE       (NPROC+MAXUSERS+(2*(100+NPROC)/3)+48) 100     46763
mtune:NDQUOT       (MAXUSERS*9+NPROC) 100     114005
mtune:HRTIME       (NPROC/4)      30      5500
mtune:HRVTIME      (NPROC/4)      30      5500
stune:NPROC        2000
```

You can use the *sysdef(1M)* command to determine the values which are currently set in the system:

```
# sysdef -i
```

Example output:

```

*
* RM600 Configuration
*
*   ...
*   ...
*
* System Configuration
*
*   rootdev      major=4(sdi)    minor=3840
*   dumpdev      major=0(nodev)  minor=0
*
*   swap files
*
*   path          dev          swaplo    blocks    free
*   /dev/ios0/sdisk01es1  4,3841      0    524288    517760
*   /dev/ios0/sdisk01ls1  4,2177      0    1638400   1634144
*   /dev/ios0/sdisk012s6  4,2310      0    4119552   4115360
*   /dev/ios0/sdisk014s7  4,2567      0    8239104   8234816
*   /dev/ios0/sdisk030s6  4,6150      0    4364288   4359840
*   /dev/ios0/sdisk031s6  4,6278      0    4364288   4359584
*   /dev/ios0/sdisk032s6  4,6406      0    4364288   4359936
*   /dev/ios0/sdisk034s5  4,6661      0    8386560   8382016
*
* Tunable Parameters
*
*   8192 buffers in buffer cache (NBUF)
*   12324 entries in callout table (NCALL)
*   12308 entries in proc table (NPROC)
*   160 Number of global scheduler priorities configured
*   99 maximum global priority in sys class (MAXCLSYSPRI)
*   1 clist buffers (NCLIST)
*   40 processes per user id (MAXUP)
*   1024 hash slots for buffer cache (NHBUF)
*   200 buffers for physical I/O (NPBUF)
*   5120 size of system virtual space map (SPTMAP)
*   0 maximum physical memory to use (MAXPMEM)
*   60 auto update time limit in seconds (NAUTOUP)
*   65536 high-water-mark of buffer cache (BUFHWM)
*   64 Number of hash buckets for proc table (NPROCHASH)
*   79 page stealing low water mark (GPGSLO)
*   2 fsflush run rate (FSFLUSHR)
*   25 minimum resident memory for avoiding deadlock (MINARMEM)
*   25 minimum swappable memory for avoiding deadlock (MINASMEM)
*   22248 maximum number of active file record locks (FLCKREC)
*
* Utsname Tunables
*
*   5.45 release (REL)
*   spinlock node name (NODE)
*   RelinatUNIX-Y system name (SYS)
*   A1010 version (VER)
*
* Process Resource Limit Tunables (Current:Maximum)
*
* Infinity :Infinity    cpu time

```

Infinity	:Infinity	file size
4000000	:4000000	heap size
80000000	:80000000	stack size
80000000	:80000000	core file size
64	:400	file descriptors
Infinity	:Infinity	mapped memory
* Streams Tunables		
* 9 maximum number of pushes allowed (NSTRPUSH)		
0 no streams threshold (STRTHRESH)		
32768 maximum stream message size (STRMSGSZ)		
1024 max size of ctl part of message (STRCTLSZ)		
* RFS Tunables		
* IPC Messages		
100 entries in msg map (MSGMAP)		
8192 max message size (MSGMAX)		
16384 max bytes on queue (MSGMNB)		
100 message queue identifiers (MSGMNI)		
8 message segment size (MSGSSZ)		
160 system message headers (MSGTQL)		
1024 message segments (MSGSEG)		
* IPC Semaphores		
100 entries in semaphore map (SEMAP)		
100 semaphore identifiers (SEMMNI)		
200 semaphores in system (SEMMNS)		
100 undo structures in system (SEMMNU)		
25 max semaphores per id (SEMMSL)		
20 max operations per semop call (SEMOPM)		
20 max undo entries per process (SEMUME)		
32767 semaphore maximum value (SEMVMX)		
16384 adjust on exit max value (SEMAEM)		
* IPC Shared Memory		
8589934592 max shared memory segment size (SHMMAX)		
1 min shared memory segment size (SHMMIN)		
2000 shared memory identifiers (SHMMNI)		
500 max attached shm segments per process (SHMSEG)		
* High Resolution Timer Tunables		
3077 max number of timer structures for real-time clock (HRTIME)		
3077 max number of timer structures for processes special clocks (HRVTIME)		
* Time Sharing Scheduler Tunables		
20 maximum time sharing user priority (TSMAXUPRI)		
SYS		system class name (SYS_NAME)
TS		class of init process (INITCLASS)

To determine the values which are set in the `/unix` kernel, proceed as follows:

```
# strings /unix | pg
```

Example output:

```
ICLR
Release: 5.45
Version: A1010
IDENT TEXT
Build/relink date: Tue Jan 20 12:54:17 MET 1999
SMP kernel
=====
.
=====
Stune:
SHMMAX 0x0000000020000000
SHMMNI 2000
SHMSEG 500
NBUF 8192
BUFHWM 65536
MAXUSERS 1536
AIO_MAX 7168
NAIOSYS 60
NRAIOBUF 1024
KV_SIZE 0x20000000
=====
Config values:
NCALL 12324
NPROC 12308
MAXUP 40
NHBUF 1024
NPBUF 200
NAUTOP 60
BDFLUSHR 2
FLCKREC 22248
PUTBUFSZ 10000
NGROUPS_MAX 16
NBUF 8192
BUFHWM 65536
PAGEIOPool_HWM 100
PAGEIOPool_LWM 50
RBUFPool_HWM 100
RBUFPool_LWM 50
ARG_MAX 20480
RSTCHOWN 1
NPROCHASH 64
NINODE 22164
UFSNINODE 22164
NDQUOT 26132
NRNODE 22164
.
.
=====
END OF IDENT TEXT
```

7.4 Modifying the kernel parameters

All kernel parameters can be modified within the range of values delimited by the minimum and maximum values specified in *mtune*. However, such modifications must not be entered in the *mtune* file but in a file named *stune* which is also located in the kernel's configuration directory */etc/conf/cf.d* (see *stune(4)*). This file does not therefore contain all the tunable parameters but only those whose values have been changed from the presetting. Some entries are performed during Reliant UNIX installation by the so-called add-on packages.



Never modify values in the file *mtune*!

You may edit the file *stune* either in order to modify the value of a kernel parameter which is already present in the file or to enter a parameter to which you wish to assign a value which is different from the default value specified in *mtune*.

RM200, RM300, RM400:

Here you can use the user interface SYSADM.

RM600:

You should use the *idtune(1M)* command. The command is a shell script located in the */etc/conf/bin* directory. *idtune* checks the new value and enters it in the *stune* file.



If, instead of this, you use an editor to process *stune* you should make sure that you do not overwrite any other entries!



When installing add-on products, check whether it is necessary to modify the values of kernel parameters. For example, databases require higher values for the semaphore and shared memory parameters.

Some kernel parameters are linked by mathematical operations. If such parameters are modified, this modification also affects the associated parameters. The following figure illustrates the kernel parameters which are affected:

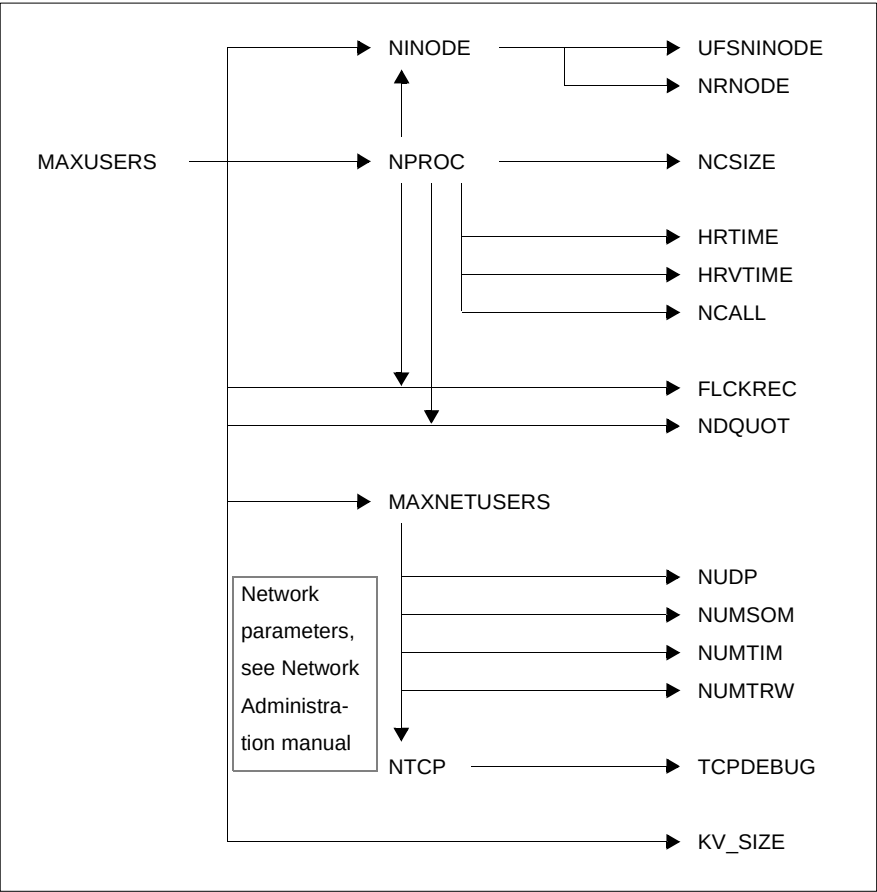


Figure 11: The interdependence of kernel parameters

This makes it clear that certain parameters have a core function, namely *MAXUSERS*, *MAXNETUSERS* and *NPROC*. For example, *MAXUSERS* affects 17 other kernel parameters. *MAXUSERS* specifies the volume of resources defined in the derived kernel parameters. By default, it has the following setting:

- 4 RM200, RM300, RM400 mit 16 Mbyte Hauptspeicher
- 8 RM200, RM300, RM400 mit 32 Mbyte Hauptspeicher
- 64 RM200, RM300, RM400 mit 64 Mbyte Hauptspeicher
- 512 RM600

7.4.1 Modifying the kernel parameters using *idtune(1M)*

The *idtune(1M)* command possesses the following syntax:

/etc/conf/bin/idtune *name value*

name the name of the parameter which is to be modified

value the new value of the parameter

Example:

```
# /etc/conf/bin/idtune NPROC 200
```

idtune searches the *mtune* file for the kernel parameter *name*. If the parameter is present then *value* is compared with the limit values. If *value* lies within the permitted range of values then *idtune* searches the *stune* file. The process continues as follows depending on whether or not the parameter is also found in this file:

- If no previous entry has been made in *stune*:

name is entered in *stune*.

- If a previous entry has been made in *stune*:

You are asked to confirm the modification.

```
Tunable Parameter "<name>" is currently set to <old value>
Is it OK to change it to <value>? (y/n) _
```

If you enter **y**, the new value is entered in *stune*. Otherwise the following message is displayed:

```
"<name>" left at <old value>
```

Assigning a parameter in *stune* the same value as its default value in *mtune* has no effect on the kernel.

7.4.2 Modifying the kernel parameters using *SYSADM*

On RM200, RM300 and RM400 systems you can modify the kernel parameters using the user interface *SYSADM*. You should proceed as follows:

- Start *SYSADM*.

The main menu of *SYSADM* appears.

- ▶ Select the menu item *performance*.

The menu *System Activity and System Tuning* appears.

- ▶ Select the menu item *tuning*.

The menu *Kernel and System Tuning* appears.

- ▶ Select the menu item *tunables*.

You will see the menu *Choice of stune* which contains all the *stune* files in the directory */opt/conf/cf.d*.

- ▶ Choose one of the *stune* files.

The following formular is displayed:

5 tuneables

TUNEABLES found in stune and mtune

Tuneable	Minimum	Value	Maximum	Input
TUNEABLES found in only in mtune				
Tuneable	Minimum	Value	Maximum	Input
AIOTIMEOUT	60	60	60	60
AIO_HIWAT	500	512	7168	512
AIO_LOWAT	20	20	20	20
AIO_MAX	500	512	7168	512
AIO_VERSION	2	3	3	3
ARG_MAX	4096	20480	51200	20480
ARPTAB_BSIZE	9	18	20	18
ARPTAB_NB	19	41	523	41
AUDITMBLK	0	0	1	0
BDFLUSHR	1	2	30	2
BEST_MATCH_ROUTING	0	1	1	1
BIGPAGESIZE	4	16	32	16

No Longer used

HELP SAVE CANCEL CMD-

Figure 12: RM200, RM300, RM400: Modifying the kernel parameters using SYSADM

The form consists of two sections:

1st section: kernel parameters that are contained in *stune* and *mtune*

2nd section: kernel parameter that are contained only in *mtune*

In each case, the kernel parameters are listed in alphabetical order. If you wish to modify a parameter you should proceed as follows:

- Position the cursor in the relevant line.

You will then be shown a brief description of this parameter in the message line. You can obtain more detailed information by pressing the function key **HELP**.

- ▶ Enter the new value.
- ▶ Finally, press the function key **SAVE**.

7.5 Generating and booting a new kernel

If you change one or more kernel parameters, the modification does not take effect immediately. First you must proceed as follows:

- ▶ generate a new kernel

RM200, RM300, RM400:

Here you can use the user interface SYSADM.

RM600:

In this case you should use the command *idbuild(1M)*. This command is a shell script which is located in the directory */etc/conf/bin*.

In either case, the initial reaction is merely to create a new file. The next time the operating system is closed down this file will be recognized and renamed to */stand/unix*.

- ▶ boot the new kernel

This boot procedure will then automatically boot the new system kernel.

7.5.1 Generating the kernel using idbuild(1M)

- ▶ Start the *idbuild* command as follows:

1-processor kernel: # /etc/conf/bin/idbuild -S

Multiprocessor kernel: # /etc/conf/bin/idbuild -M

Generation takes a certain time. While this is happening the following or similar messages may be displayed:

```
The UNIX Operating System will now be rebuilt.  
This will take some time. Please wait.  
  
idaddsyms: 8464 symbols (135424 bytes), 98121 bytes of strings  
  
The UNIX Kernel has been rebuilt.
```

7.5.2 Generating the kernel using SYSADM

On the RM200, RM300 and RM400 you can generate the system kernel using the user interface SYSADM. You should proceed as follows:

- ▶ Start SYSADM.
The main menu of SYSADM appears.
- ▶ Select the menu item *performance*.
The menu *System Activity and System Tuning* appears.
- ▶ Select the menu item *tuning*.
The menu *Kernel and System Tuning* appears.
- ▶ Select the menu item *build*.
The menu *build* appears.
- ▶ Select the menu item *idbuild*.
The following formular is displayed:

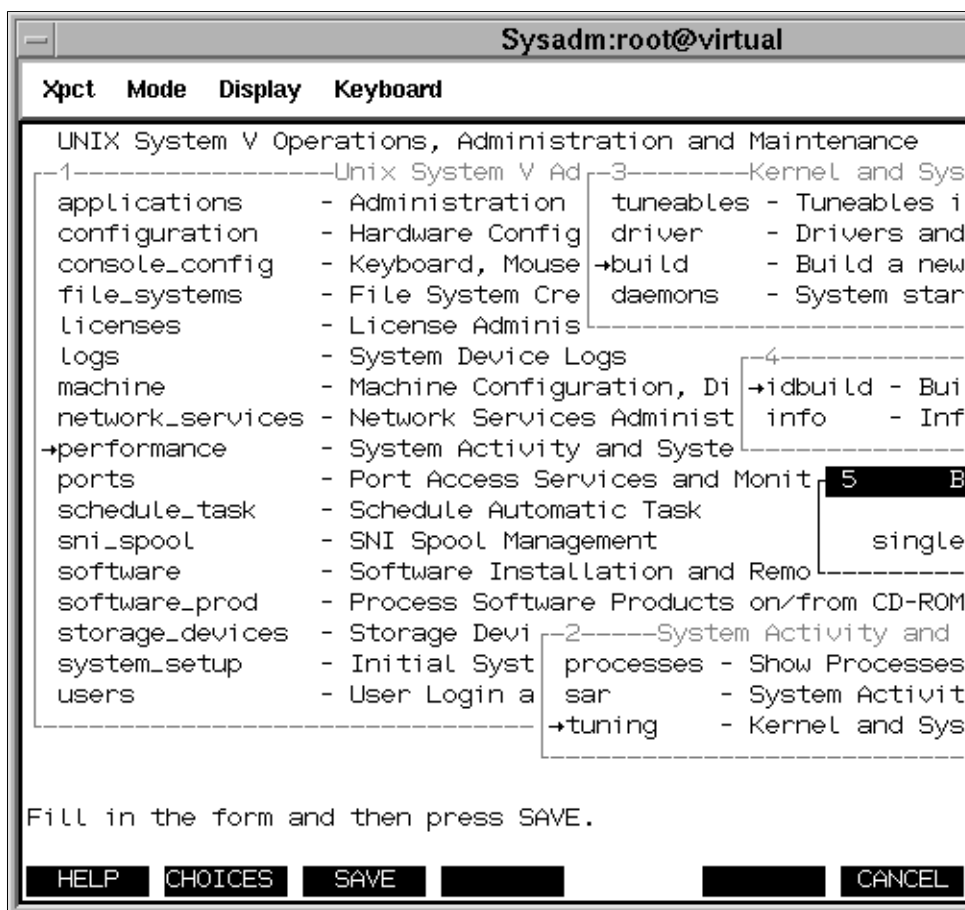


Figure 13: RM200, RM300, RM400: Generating a new kernel using SYSADM

- ▶ Specify whether a single-processor or multiprocessor kernel should be built.
Use the function key **[CHOICES]** for this.
- ▶ Then press the function key **[SAVE]**.
If you wish to delete the newly built kernel, you can do this from the menu item *info* in the menu *build*.
- ▶ Then exit from SYSADM.

7.5.3 Booting the new kernel



Multiuser mode:

Before powering down the application, check that no other users are still signed on (e. g. using the *who(1)* command). If there are, you should inform these other users about the pending change in operating status and give them enough time to finish their work. If you initiate the boot process with the *shutdown(1M)* command, all users who are currently signed on will be informed automatically.

- Change to the */* directory and boot the new kernel.

```
# cd
# /etc/shutdown -i6 -g0 -y
```

-i6 Powers up the system to run level 6

-g900 Sets the time for logging off to 900 seconds (i.e. 15 minutes)

The new kernel is activated on power down. You will see the message:

```
setting up new kernel as next default kernel: done.
```

For detailed information on powering the operating system up and down, refer to the manual “Reliant UNIX Operation”.

7.6 Booting the old kernel

There are only very few cases in which the new kernel cannot be booted correctly. However, this may occur if, for example, you have modified a kernel parameter or combination of parameters in such a way that the new kernel is too large or cannot be correctly initialized. In such an event you must first boot the old kernel in order to obtain a usable operating system. This kernel will have been renamed to */stand/unix.old* when Reliant UNIX was powered down for the last time. To boot the old kernel, proceed as follows:

RM200, RM300, RM400

- Switch to the Prom Monitor / Firmware Monitor.
- Boot the SASH (Stand-Alone SHell).

Prom Monitor: >> boot

Firmware Monitor: Stand-alone shell boot (boot)

► Boot the old kernel.

- sash: `dkncr(c,t,s)unix.old` internal system disk
- sash: `dkpcs(c,,t,s)unix.old` external system disk (SNI SCSI controller)
- sash: `dkdpt(c,,t,s)unix.old` external system disk (DPT SCSI controller)

The variables *c*, *t*, *s* represent the following values:

- c* number distributed from the firmware
- t* SCSI ID
- s* number of the partition

RM600-xxx, RM600 E

- Switch to the Board Debug Monitor (BDM).
- Boot Boot2.

Specify the boot string of the active system disk. Example:

Logical boot mode BDM(SPB 8 t)> b 201 sdisk011s0/sash

Physical boot mode BDM(SPB 8 t)> b 201 sd(1,8,0)sash

- Boot the old kernel.

```
boot2: boot -f sd(,,)unix.old
```

Saving a usable kernel

- When UNIX reports, save the usable kernel in case the next version also fails to function.

```
# mv /stand/unix.old /stand/unix.sav
```

- Restore the kernel parameters in */etc/conf/cf.d/stune* to their original values (using the *idtune* command).
- Execute the *idbuild* command again.
- Reboot UNIX.

If, for whatever reason, this kernel should also fail to boot, boot from *unix.sav* instead of *unix.old* and try again.



Each kernel occupies approximately 9 Mbytes in the directory */stand!*

7.7 The rlimit structure

For system resources, the *rlimit* structure defines “hard” (absolute) and “soft” (adjustable) limits (see section on page 198).

These limits are passed to the 0 process, after which each child process inherits the limits of the parent process. However, if a process uses *exec* to executes a file whose set *userid* bit (owner or group) is set then the resource limit values of this process are reset to their default values.

All processes can change their own limit values by using *setrlimit* (see *getrlimit(2)*). They can increase their soft limit up to but not beyond the value of the hard limit. Only processes whose effective user ID is equal to 0 (*root*) can increase their hard limit. In contrast, all processes can lower their hard limit. If a limit value is equal to 0x7FFFFFFF (32 bit kernel) or 0x7FFFFFFFFFFFFFFF (64 bit kernel) (*RLIMIT_INFINITY*) then there is no upper limit for this resource.

7.8 Description of the kernel parameters

This section describes all the relevant parameters which are described in the file */etc/conf/cf.d/mtune*. The parameters are listed by parameter type (for an overview, see page 164). The section of *mtune* in which the parameter can be found is also specified (specification SECTION *number*).



The default values specified for the kernel parameters below are valid for Reliant UNIX 5.45 running on an RM600. Most of these values also apply to other RM computers. However, in individual cases differences may occur. Please consult the *mtune* file to ascertain the default values which apply in your system.

Many parameters are only used by device drivers or other kernel components which form part of an add-on package. If no such add-on packages are installed then modifying the values of these parameters has no effect on the system configuration.

The names of the kernel parameters share a uniform syntax. Some name components indicate the parameter type:

Abbreviation	Parameter type	Examples
<i>LIM</i>	Resource limit parameters	<i>SCPULIM, HCPULIM</i>
<i>MSG</i>	Message parameters	<i>MSGMAP, MSGMAX</i>
<i>SEM</i>	Semaphore parameters	<i>SEMMAP, SEMMNI</i>
<i>SHM</i>	Shared memory parameters	<i>SHMMAX, SHMMIN</i>
<i>STR</i>	STREAMS parameters	<i>NSTRPUSH, STRMSGSZ</i>

Other name components are abbreviations. The table below contains a number of abbreviations which either occur frequently or are highly typical.

Abbreviation	Meaning	Examples
<i>BUF</i>	buffer	<i>NHBUF, NPBUF</i>
<i>H</i>	hard limit	<i>HCPULIM, HFSZLIM</i>
<i>HWM</i>	high water mark	<i>PAGEIOPOOL_HWM</i>
<i>LWM</i>	low water mark	<i>PAGEIOPOOL_LWM</i>
<i>MAX</i>	maximum	<i>MAXUP, MAXUSERS</i>
<i>MEM</i>	memory	<i>MAXPMEM, MINARMEM</i>
<i>MIN</i>	minimum	<i>MINARMEM, SHMMIN</i>
<i>N</i>	number	<i>NCALL, NPROC</i>
<i>S</i>	soft limit	<i>SPCULIM, SFSZLIM</i>

7.8.1 General kernel parameters

<i>ARG_MAX</i>	Default value: 20480	SECTION 2
Maximum number of characters (including the NULL character) which can be used in the <i>exec</i> call.		
<i>BDFLUSHR</i>	Default value: 2	SECTION 2
<i>BDFLUSHR</i> specifies the number of seconds after which the <i>fsflush</i> process is called in order to write a part of the data buffers which have been modified to the file systems on the hard disk. The default values (<i>NAUTOUP</i> = 60 and <i>BDFLUSHR</i> = 2) mean that each time the <i>fsflush</i> process is called, 1/30 of the data buffers are checked. This does not represent a full update of the file systems but serves to avoid peak I/O load during <i>sync</i> (see also <i>NAUTOUP</i>).		
<i>BUFHWM</i>	Default value: 256	SECTION 2
Maximum size of the “old” buffer cache in units of 16 Kbytes. The buffer cache is used for file system management (inodes, indirect blocks and file system management blocks). When the system is started, a check is performed to determine whether the size of the buffer cache lies within certain limit values. If necessary, the size of the buffer cache is adjusted as appropriate (see page 12).		
<i>FLCKREC</i>	Default value: $((MAXUSERS + NPROC + 16) * 16 / 10 + 72)$	SECTION 2
Maximum number of locked files or records. These entries for file and record locks may be dynamically allocated if necessary (see also <i>fcntl(2)</i>).		
<i>MAXMINOR</i>	Default value: 0x3FFFF	SECTION 3
Maximum device number (minor) which can be assigned to a device node which is generated using <i>mknod(2)</i> . The default value is also the maximum value for this parameter.		
<i>MAXPMEM</i>	Default value: 0	SECTION 3
Maximum size of the physical memory (in pages) which is available for use. The default value 0 specifies that the entire main memory is available for use. This parameter should not be modified.		

MAXUP Default value: 40 SECTION 2

Maximum number of processes which a user may use simultaneously. This number applies per user ID and not per terminal. If, for example, two users are logged on with the same user ID, the upper limit will be quickly reached.

NAUTOUP Default value: 60 SECTION 2

NAUTOUP specifies the maximum number of seconds after which the file systems on the hard disk must be updated with the information from the operating system's data buffers (see also *sync(2)*).

Reducing *NAUTOUP* (i. e. increasing the *sync* rate) reduces the risk of losing data in the event of a system crash. However, this also reduces system performance (and vice versa).

NBUF Default value: 512 SECTION 2

The data buffers of the buffer cache are managed in so-called buffer heads. When the operating system is started, *NBUF* buffer heads are reserved. If no more buffer heads are available for buffer management then the system dynamically allocates additional buffer blocks.

There is no upper limit to the number of buffer heads. The *BUFHWM* parameter limits the number of Kbytes which are available to data buffers. This limits the number of buffer blocks which can be allocated.

The dynamically allocated buffer heads are returned to the memory management when they are no longer needed. Setting a higher value for *NBUF* reduces the frequency with which the kernel's memory allocation routine has to be called in order to reserve space for the buffer heads. However, this also results in more memory being made unavailable for other purposes in the system address space.

NCALL Default value: (*NPROC* + 16) SECTION 2

Maximum number of callout structures. These structures contain information about functions which are to be started by a routine which is controlled by the system clock. The range of values for this parameter is between 30 and 21861.

NC_NAMLEN Default value: 25 SECTION 3

Maximum length of component names which are saved in the DNLC.

<i>NCSIZE</i>	Default value: $(NPROC + 100)$ if $(NPROC + 100) < 2048$ 2048 if $(NPROC + 100) \geq 2048$	SECTION 3
Maximum number of entries in the DNLC (Directory Name Lookup Cache). The DNLC contains the names of all the files and directories which have been used during the processing of pathnames. Pathname processing is accelerated if the names of the components are found in the DNLC. If necessary, the least frequently used entry may be replaced by a new entry. The DNLC should be considerably smaller than the inode cache (<i>NINODE</i>), so that it does not constantly use up all of the inode cache entries.		
<i>NGROUPS_MAX</i>	Default value: 16	SECTION 2
Number of user groups for which a user may possess permissions. All users may possess multiple group permissions at any given time without having to execute the <i>newgrp</i>(1) command (see <i>getgroups</i>(1)).		
<i>NHBUF</i>	Default value: 1024	SECTION 2
Number of entries in the hash table of the buffer cache. The hash table is used to locate the associated data buffer via the device and block numbers (of a file's data block). This procedure is faster than a linear search through the entire buffer list. The value of <i>NHBUF</i> must be a power of 2.		
<i>NPBUF</i>	Default value: 200	SECTION 2
Maximum number of buffers which are used for physical input/output. One input/output buffer is required for each physical read/write operation. If many such jobs are performed in parallel, then the number of buffers should be increased, since processes wait until these buffers are freed. In general, the greater the number of configured hard disks, the larger the number of buffers which are required. Streamers and CD-ROM's also use these buffers.		

NPROC

Default value: $(MAXUSERS * 8 + 20)$

SECTION 2

Maximum size of the process table. The process table is managed dynamically. It is only ever as large as the number of active processes. However, *NPROC* limits the number of processes which can run simultaneously within the system. Normally, the more users there are, the greater the number of processes. If insufficient processes are permitted, the factor value 8 should be increased.

Although the process table is dynamic, there are also static data fields (e.g. semaphores) which use this constant for dimensioning purposes. The number of processes which users are allowed to generate is also limited (see *MAXUP*). The maximum value of this parameter is 12308.

NPROCHASH

Default value: 64

SECTION 2

The number of hash entries which are used to search through the *proc* table. Processes are entered in a hash list in order to facilitate the rapid location of processes which corresponds to certain process IDs (PID). A large list results in rapid process location but consumes more memory. The value of this parameter must be a power of 2. If you enter a different value you get the message *v.v_phash* adjusted to *number*.

PAGEIOPOOL_HWM

Default value: 100

SECTION 2

PAGEIOPOOL_LWM

Default value: 50

SECTION 2

The paging procedure requires special job structures for input/output. *PAGEIOPOOL_LWM* defines the number of such structures which are reserved for the paging procedure at system start. If, during operation, more than *PAGEIOPOOL_LWM* job structures are needed, then additional job structures are reserved, providing that the value of *PAGEIOPOOL_HWM* has not yet been reached. If more than *PAGEIOPOOL_HWM* job structures are required then the excess structures are dynamically allocated and subsequently released when they are no longer needed.

PUTBUFSZ

Default value: 10000

SECTION 2

Size of the ring buffer *putbuf*. Thanks to the *putbuf* ring buffer it is possible, following a system crash, to consult a dump of the error messages which were output at the console. This buffer is written cyclically, i.e. after *PUTBUFSZ* characters, old messages are overwritten. To view the contents of this buffer, use *crash(1M)* and *ikdb(8)*.

<i>RBUFPOOL_HWM</i>	Default value: 100	SECTION 2
<i>RBUFPOOL_LWM</i>	Default value: 50	SECTION 2

Number of job structures which are reserved for raw devices and virtual disks at system start. If, during operation, more than *RBUFPOOL_LWM* job structures are needed, then additional job structures are reserved, providing that the value of *RBUFPOOL_HWM* has not yet been reached. If more than *RBUFPOOL_HWM* job structures are required then the excess structures are dynamically allocated and subsequently released when they are no longer needed.

<i>RSTCHOWN</i>	Default value: 1	SECTION 2
-----------------	------------------	-----------

This parameter specifies who may modify a file's ownership rights. Only the values 0 and 1 are permitted for *RSTCHOWN*:

With the value 0, the owner of a file can change the user ID and group ID of a file to any desired value, including non-existent group and user IDs.

The value 1 corresponds to FIPS/BSD mode (Federal Information Processing Standards/Berkeley Software Department). In this mode, the options for modifying a file's ownership rights are restricted. Only *root* and processes running with *root* permissions (all processes whose user ID is 0) may modify the ownership rights and group ID of a file at will. The owner of a file may only change a file's group ID to the ID of another group to which the owner belongs (see *getgroups(1)*).

<i>SDILARGEIO</i>	Default value: 256	SECTION 8
-------------------	--------------------	-----------

This parameter allows inputs/outputs larger than 64 Kbytes for newer disks (e.g. MP48, MP98), thus enhancing performance

<i>SHLBMAX</i>	Default value: 3	SECTION 3
----------------	------------------	-----------

Maximum number of sharable static libraries which a process may access simultaneously. Reliant UNIX also supports the shared use of dynamic libraries. *SHLBMAX* only affects the static, sharable libraries of old applications which use the COFF format (COFF = Common Object File Format). You should only modify this parameter if an *ELIBMAX* error is reported (Attempting to link in more shared libraries than system limit).

SPTMAP

Default value: 5120

SECTION 3

Size of the field in which the dynamically allocated virtual addresses of the kernel are managed.

7.8.2 File system parameters

NDQUOT

Default value: $(MAXUSERS * 9 + NPROC)$

SECTION 2

Size of the kernel's quota table for *UFS* type file systems. For all users this must contain an entry which specifies a quota for each mounted *UFS*. If quotas are in use, the table entries limit the amount of disk storage which is available to a user. If no entries are available, the message `dquot table full` is output at the console. This may occur if too many users are working at the computer. If this happens, you should increase the value of *NDQUOT*.

NINODE

Default value:
 $(MAXUSERS + NPROC + (2 * (100 + NPROC) / 3) + 48)$

SECTION 2

This parameter is no longer directly used by the kernel. It is now simply used to calculate the parameters *UFSNINODE* and *NRNODE*. The Veritas inode cache is also dimensioned on the basis of *NINODE*.

NRNODE

Default value: $(NINODE)$

SECTION 2

Maximum number of *rnode* structures which can be allocated to an NFS (Network File System).

UFSNINODE

Default value: $(NINODE)$

SECTION 2

Maximum number of files, directories and device files belonging to file systems of the type *UFS* (UNIX File System) which can be used simultaneously. The term “used” applies, for example to inodes with *DNLC* entries and open files. One inode entry is created in the system memory for each used file, directory and device file belonging to a *UFS*.

7.8.3 **Paging parameters**

<i>GPGSLO</i>	Default value: -1	SECTION 2
Lower limit of free memory (in pages) at which the swap daemon starts to swap out processes. If you want to increase daemon activity, you should increase this value. However, if you want to reduce activity, you should reduce this value. The default value of -1 means that the value is automatically determined.		
<i>MINARMEM</i>	Default value: 25	SECTION 3
Minimum number of main memory pages which must be available for the paging procedure.		
<i>MINASMEM</i>	Default value: 25	SECTION 3
Minimum number of main memory and swap pages which must be reserved to prevent deadlocks.		
<i>PAGEOUT_DELTA</i>	Default value: 0	SECTION 3
The <i>pageout</i> daemon normally stops working when the number of free main memory pages is again equal to <i>LOTSFREE</i> . However, if <i>PAGEOUT_DELTA</i> is greater than 0, the daemon continues to operate until an additional proportion of paging address space is free. This proportion is calculated as follows: $(PAGEOUT_DELTA / 100) * \text{size of the available paging address space}$.		
You should only modify this parameter if sufficient real main memory is available for the processes which are simultaneously running on the system.		

7.8.4 **STREAMS parameters**

The following parameters are used for processing STREAMS.

<i>NAUTOPUSH</i>	Default value: 64	SECTION 2
Number of devices for which a STREAM module can automatically be entered. If the <i>ioctl(2)</i> system call with the parameter <i>SAD_SAP</i> (see <i>sad(7)</i>) returns an <i>ENOSR</i> error (Out of stream resources) you should increase the value of this parameter.		

NLOG Default value: 10 SECTION 2

The number of devices which are available via the clone interface of the log driver (*/dev/log*). If */dev/log* cannot be opened and an *ENXIO* error is reported (No such device or address), then it may be necessary to increase this value.

NSTRPHASH Default value: 64 SECTION 2

Size of the internal hash table. You will probably never have to change this parameter unless your system uses a very large number of drivers.

NSTRPUSH Default value: 9 SECTION 2

Maximum number of modules which can be packed in a STREAM. This parameter is used to prevent a defective user process from using all the available queues in a single STREAM.

NUMSAD Default value: 8 SECTION 2

No. of devices available via the STREAMS Administration Driver (SAD).

STRCTLSZ Default value: 1024 SECTION 3

Maximum permitted size of the monitoring or control information in a STREAM message. The control information in a *putmsg(2)* message is not subject to the maximum and minimum packet size restrictions. Such control information is limited by the *STRCTLSZ* parameter alone. The default value of 1024 is generally sufficient for existing applications.

STRMSGSZ Default value: 32768 SECTION 2

Maximum permitted size of the data field in a STREAM message. This parameter should normally be large enough to cater for the maximum packet size restrictions of all the configured STREAM modules. The value 0 means that there is no upper limit.

<i>STRTHRESH</i>	Default value: 0	SECTION 3
Maximum memory in bytes which STREAM modules may occupy. The value 0 means that there is no upper limit. If this limit is exceeded then users who do not possess the corresponding access permissions may not open any new STREAMS, pack modules on a STREAM or write to STREAM devices. Such users see the <i>ENOSR</i> error message (Out of stream resources). Users who possess the appropriate access permissions may always ignore this message.		

7.8.5 Message parameters

The message parameters affect interprocess communication messages (IPC parameters, IPC = InterProcess Communication).

<i>MSGMAP</i>	Default value: 100	SECTION 2
Size of the control field used to manage message segments.		
<i>MSGMAX</i>	Default value: 8192	SECTION 2
Maximum size of the messages of the <i>msgsnd</i> and <i>msgrcv</i> commands (parameter: <i>size_t msgsz</i>, see <i>msgop(2)</i>). If greater data volumes are required, you must increase the value of <i>MSGMAX</i> accordingly. The default value is normally sufficient since long messages are broken down into smaller units by the user process.		
<i>MSGMNB</i>	Default value: 16384	SECTION 2
Default size for <i>msg_qbytes</i> which is made available when <i>msgget(2)</i> is called. This value can be modified using the <i>msgctl(2)</i> system call (<i>cmd</i> control function <i>IPC_SET</i>) after creation of the message queue (by the user). <i>msg_qbytes</i> is the maximum number of bytes which are reserved for the message queue. If, on transmission, the length of a message exceeds the length of the queue, the message is blocked until there is again space in the queue. Exception: If <i>msgflg&IPC_NOWAIT</i> is set, then <i>msgsnd</i> is terminated immediately, see <i>msgsnd(2)</i>. The parameter should be larger than <i>MSGMAX</i> as under certain circumstances it may otherwise impossible to send even the first message.		

<i>MSGMNI</i>	Default value: 100	SECTION 2
Number of message queues (Id structure) for the entire system. If two processes communicate via messages, then messages awaiting processing are stored in so-called message queues.		
<i>MSGSEG</i>	Default value: 1024	SECTION 2
Number of message segments in the system. The product of multiplying this value by that of <i>MSGSSZ</i> must be smaller than 4 Mbytes.		
<i>MSGSSZ</i>	Default value: 8	SECTION 2
Size of a message segment in bytes. Messages consist of a sequence of message segments which are large enough to contain the entire text. The value of <i>MSGSSZ</i> must satisfy two conditions: it must be a power of 2 and the product of multiplying this value by that of <i>MSGSEG</i> must be smaller than 4 Mbytes.		
<i>MSGTQL</i>	Default value: 160	SECTION 2
Number of message headers in the system and, consequently, also the number of outstanding messages. Message headers contain message management information.		

7.8.6 Semaphore parameters



When installing add-on products, note whether it is necessary to modify the value of any kernel parameters. For example, databases require high values for semaphore and shared memory parameters.

<i>SEMAEM</i>	Default value: 16384	SECTION 3
Maximum value of a <i>sem_undo</i> entry. This parameter cannot be modified.		
<i>SEMMAP</i>	Default value: 100	SECTION 2
Size of the control field for semaphore groups.		
<i>SEMMNI</i>	Default value: 100	SECTION 2
Maximum number of semaphore groups. Each semaphore group is identified by a unique number. When <i>SEMMNI</i> numbers have been allocated, no further semaphores can be used.		

<i>SEMMNS</i>	Default value: 200	SECTION 2
Maximum number of possible semaphores. A semaphore group consists of a number of individual semaphores. Both the number of semaphores and the number of semaphore groups is limited.		
<i>SEMMNU</i>	Default value: 100	SECTION 2
Number of <i>sem_undo</i> structures in the system. This value is equal to $8 * (SEMUME + 2)$ bytes. If the <i>semop(2)</i> system call is used, a flag can be set at <i>SEM_UNDO</i> . In this case one <i>sem_undo</i> structure is reserved. <i>SEMMNU</i> limits the number of such structures present in the system.		
<i>SEMMSL</i>	Default value: 25	SECTION 2
Maximum number of semaphores in a group (i. e. per semaphore ID).		
<i>SEMOPM</i>	Default value: 20	SECTION 2
Maximum number of semaphore operations which can be performed per <i>semop(2)</i> system call. A semaphore operation may modify or read a semaphore. 20 simultaneous operations can therefore be performed on the semaphore group in a single <i>semop</i> call.		
<i>SEMUME</i>	Default value: 20	SECTION 2
Maximum number of undo operations per undo structure.		
<i>SEMVMX</i>	Default value: 32767	SECTION 3
Maximum value a semaphore can have. This parameter cannot be modified.		

7.8.7 Shared memory parameters



When installing add-on products, note whether it is necessary to modify the value of any kernel parameters. For example, databases require high values for semaphore and shared memory parameters.

<i>SHMMAX</i>	Default value: 0x0000000200000000	SECTION 2
Maximum size (in bytes) of a requested shared memory segment.		

SHMMIN Default value: 1 SECTION 3

Minimum size (in bytes) of a requested shared memory segment. This parameter cannot be modified.

SHMMNI Default value: 100 SECTION 2

Number of unique identifiers for shared memory segments (and consequently also the number of differently allocated shared memory segments).

SHMSEG Default value: 20 SECTION 2

Maximum number of shared memory segments which can be allocated to a process. The number of shared memory segments which can actually be allocated to a process also depends on other factors (e.g. the *SVMLIM* parameter or the available swap area). Even if a process possesses fewer than *SHMSEG* shared memory segments it is possible that it will be allocated no additional memory because none is available.

7.8.8 Device driver parameters

NCPYRIGHT Default value: 10 SECTION 3

Size of a kernel data structure which is required in order to output the initialization messages of certain device drivers (e.g. *lce*) at the console. This parameter cannot be modified.

NUMSXT Default value: 6 SECTION 3

Number of drivers for available shell layers. This parameter is not used in the kernel.

PRFMAX Default value: 32768 SECTION 3

Maximum number of text symbols which the kernel's profile generation routine (*/dev/prf*) can process correctly (see *prf(7)* and *profiler(1M)*).

7.8.9 Timer and scheduler parameters

HRTIME Default value: (*NPROC* / 4) SECTION 3

Number of *hrtimes* structures. These structures monitor interrupt and alarm requests from the real-time clock. Each process can request multiple “High Resolution Timers”. If all the *hrtimes* structures have been allocated then the *getitimer*(3C) and *setitimer*(3C) functions report an error.

HRVTIME Default value: (*NPROC* / 4) SECTION 3

Number of *itimes* structures. These structures monitor the alarm requests for clocks which measure the virtual times of processes. Each process can request multiple “High Resolution Timers” for the measurement of virtual computation times.

Virtual time means that the process does not receive a signal until it itself has consumed the computation time. Logically, this takes longer when many processes are running than when there are only a small number of processes. The function is used to measure computation time.

MAXCLSYSPRI Default value: 99 SECTION 3

Maximum global priority which can be assigned to system processes. Modifying this parameter also results in the modification of the priority range for system processes. This can significantly influence system performance. In general it is not necessary to modify *MAXCLSYSPRI*. An exception occurs when you add new scheduler classes or modify the priority of currently configured classes.

TPCLOCK Default value: 1 SECTION 3

Number of pulses of the high resolution timer during clock processing. This parameter cannot be modified.

TSMAXUPRI Default value: 20 SECTION 2

The range within which a user can modify the user priority of a timesharing process lies between *-TSMAXUPRI* and *+TSMAXUPRI*. If you increase the value of *TSMAXUPRI* you can control process priorities better. (Only *root* can increase priorities in all cases.) The default value of 20 corresponds to the control level provided by the *nice*(1) interface.

<i>TSSLEEPWAIT</i>	Default value: 4	SECTION 2
<i>TSSLEEPDECAY</i>	Default value: 2	SECTION 2
<i>TSSLEEPMAX</i>	Default value: 60	SECTION 2

These parameters relate to the process scheduler of the timesharing mode. *TSSLEEPWAIT* is the minimum time that an inactive process must wait before its priority is increased. When this period has elapsed, the process is moved one priority class higher. It is then assigned a new wait time which is shorter than the previous wait time by the factor $1/(2^{TSSLEEPDECAY})$. If the previous wait time was larger than *TSSLEEPMAX* then *TSSLEEPMAX* is specified for the recalculation of the wait time instead of the old wait time.

<i>TSSLPSWPDIV</i>	Default value: 20	SECTION 2
--------------------	-------------------	-----------

Sensitivity used when checking whether a sleeping process should be swapped out of main memory into the swap area. When the minimum value of 10 is set, sensitivity is greatest. It is then more likely that a process which has been sleeping for a long period will be swapped out than when the value of *TSSLPSWPDIV* is high.

7.8.10 Resource limit parameters

The following parameters are default values which determine process resource limits. These parameters are needed to make entries in the *rlimit* structure. In Reliant UNIX, the *ulimit(1)* command provides users with a number of new kernel limits which are taken from the *rlimit* structure. In the parameters below, the *S* prefix means that this is a soft limit and the *H* prefix that the parameter represents a hard limit. All user processes are able to increase their soft limits to the value of their hard limits. Only *root* is able to increase the hard limits. Setting a parameter to 0x7FFFFFFF (32 bit kernel) or 0xFFFFFFFFFFFFFFFF (64 bit kernel) means that there is no upper limit, that is to say that the value may theoretically be infinite.

<i>SCORLIM</i>	Default value: 0x80000000	SECTION 2
----------------	---------------------------	-----------

Maximum size (in bytes) of a *core* file (soft limit). If this parameter is set to 0x0 only empty *core* files are created.

<i>HCORLIM</i>	Default value: 0x80000000	SECTION 2
----------------	---------------------------	-----------

Maximum value of *SCORLIM*.

<i>SCPULIM</i>	Default value: 0x7FFFFFFFFFFFFFFF	SECTION 2
Maximum computation time (in seconds) which is allocated to a process (soft limit). The computation time of a process consists of the user and system time (see also <i>time(1)</i>). Once this preset time has elapsed, the process receives a <i>SIGXCPU</i> signal. It is then generally terminated.		
<i>HCPULIM</i>	Default value: 0x7FFFFFFFFFFFFFFF	SECTION 2
Maximum value for <i>SCPULIM</i> .		
<i>SDATLIM</i>	Default value: 0x4000000	SECTION 2
Maximum heap size (in bytes) of a process (soft limit). If a process attempts to extend its heap beyond this limit using <i>brk(2)</i> then the attempt fails. An <i>ENOMEM</i> error is reported (Not enough space).		
<i>HDATLIM</i>	Default value: 0x4000000	SECTION 2
Maximum value of <i>SDATLIM</i> .		
<i>SFNOLIM</i>	Default value: 100	SECTION 2
Maximum number of files which a process can have (soft limit). If a process attempts to open another file once this limit has been reached, the attempt fails. An <i>EMFILE</i> error is reported (Too many open Files). The parameter <i>SFNOLIM</i> replaces the SVR3.2 parameter <i>NOFILE</i> .		
<i>HFNOLIM</i>	Default value: 0x400	SECTION 2
Maximum value of <i>SFNOLIM</i> .		
<i>SFSZLIM</i>	Default value: 0x7FFFFFFFFFFFFFFF	SECTION 2
Maximum permitted size (in bytes) which may be attained by a file which is created or enlarged by a process (soft limit). If a process attempts to write a file which is larger than <i>SFSZLIM</i> , a <i>SIGXFSX</i> signal is sent and the write operation is aborted with the <i>EFBIG</i> error message (File too large).		
<i>SFSZLIM</i> replaces the kernel parameter <i>ULIMIT</i> from SVR3.2. It is also possible to disable this kernel limit at user level, for example by using the shell command <i>ulimit(1)</i> or the <i>ULIMIT</i> entry in <i>/etc/default/login</i> .		

<i>HFSZLIM</i>	Default value: 0x7FFFFFFFFFFFFFFF	SECTION 2
Maximum value of <i>SFSZLIM</i> .		
<i>SSTKLIM</i>	Default value: 0x80000000	SECTION 2
Maximum size (in bytes) of the stack area of a process (soft limit). This parameter defines the limit for the automatic stack area in the system. If the stack size of a process exceeds this value then a <i>SIGSEGV</i> signal is sent to the process. The process is terminated at this point unless it has a way of intercepting this signal in another stack (see <i>sigaltstack(2)</i>).		
<i>HSTKLIM</i>	Default value: 0x80000000	SECTION 2
Maximum value of <i>SSTKLIM</i> .		
<i>SVMLIM</i>	Default value: 0x7FFFFFFFFFFFFFFF	SECTION 2
Maximum address space which may be allocated to a process (soft limit). Attempts to increase the address space of a process beyond this limit (i. e. with <i>brk(2)</i> , <i>shmat(2)</i> and <i>mmap(2)</i>), result in an <i>ENOMEM</i> error (Not enough space).		
<i>HVMLIM</i>	Default value: 0x7FFFFFFFFFFFFFFF	SECTION 2
Maximum value of <i>SVMLIM</i> .		

7.8.11 Specific parameters in Reliant UNIX

General kernel parameters

<i>AIO_HIWAT</i>	Default value: 512	SECTION 3
<i>AIO_HIWAT</i> defines the maximum number of free operation structures that are firmly reserved for asynchronuos I/O when they have been assigned once. This parameter does not need to be changed even if <i>AIO_MAX</i> is increased.		
<i>AIO_LOWAT</i>	Deafult value: 20	SECTION 3
<i>AIO_LOWAT</i> has been introduced to maintain compatibility. It defines the minimum number of operation structures that is always reserved. This parameter should not be changed.		

AIO_MAX

Deafult value: 512

SECTION 3

AIO_MAX defines the maximum number of asynchronous I/O operation structures that can be assigned dynamically in the operating system. Each asynchronous I/O operation takes up exactly one operation structure.

The following message is returned when the maximum number has been reached, depending on the interface used: *EWOULDBLOCK* (*specaio* interface), *EAGAIN* (*ISTD* interface), *ENOMEM* (*ABI AIO* interface).

If an application requires more than the default number of concurrent asynchronous I/O operations, *AIO_MAX* can be increased.

AIO_VERSION

Deafult value: 3

SECTION 3

The parameter determines the version number which is returned from the *DKIOCAIOVERS* function of the *ioctl* system call.

If old applications expect the former value 2 then *AIO_VERSION* can be set to 2. Wherever possible, though, applications should be modified to look for 3 and be recompiled.

- KM_CACHE_SMALL*

Default value: 64

SECTION 3

for memory requests of 1 - 512 bytes
- KM_CACHE_LARGE*

Default value: 16

SECTION 3

for memory requests of 513 bytes - 8 Kbytes
- KM_CACHE_OSIZE*

Default value: 8

SECTION 3

for memory requests over 8 Kbytes
- KM_CACHE_FAST*

Default value: 32

SECTION 3

For fast memory requests

In order to accelerate access to the system kernel memory, the memory management uses local CPU cache lists. The maximum number of buffers in the cache lists is defined by *KM_CACHE_**.

<i>KM_OSB_HWM_0</i>	Default value: 16	SECTION 3
<i>KM_OSB_HWM_1</i>	Default value: 8	SECTION 3
<i>KM_OSB_HWM_2</i>	Default value: 4	SECTION 3
<i>KM_OSB_HWM_3</i>	Default value: 2	SECTION 3
<i>KM_OSB_HWM_[4-7]</i>	Default value: 0	SECTION 3

KM_OSB_HWM_[0-7] are used to define the maximum number of free buffers in the respective free lists (see *KM_OSB_SIZE_**).

<i>KM_OSB_SIZE_0</i>	Default value: 16*1024	SECTION 3
<i>KM_OSB_SIZE_1</i>	Default value: 32*1024	SECTION 3
<i>KM_OSB_SIZE_2</i>	Default value: 48*1024	SECTION 3
<i>KM_OSB_SIZE_3</i>	Default value: 64*1024	SECTION 3
<i>KM_OSB_SIZE_[4-7]</i>	Default value: 0	SECTION 3

Up to 8 free lists can be configured for oversized memory requests (> 8 Kbytes). The size of this type of list is specified as a multiple of the page size applicable in the system (16 Kbytes). A free list is only configured for entries other than 0.

<i>KV_SIZE</i>	Default value:	SECTION 3
(((<i>MAXUSERS</i> - 64) / 41 + 8) * 0x200000)		

Size of the virtual memory which can be dynamically allocated to the kernel.

<i>NPTM</i>	Default value: 1024	SECTION 2
Number of usable ptem structures in the kernel (ptem = Pseudo Terminal Emulation Module). ptem is a STREAMS module. The value of this parameter should be greater than or equal to <i>NPTY</i> . It should be increased if the following error is reported at the console: No ptem structures. See also <i>ptem(7)</i> .		

<i>NPTY</i>	Default value: 1024	SECTION 2
Number of possible pseudo-TTYs. Pseudo-TTYs are used, for example, when a window is opened at an X terminal. This is done by using the pseudo-devices from the <i>/dev/pts</i> directory. If you attempt to open more than <i>NPTY</i> structures then no error is reported at the console. The corresponding "open" function simply returns an <i>ENODEV</i> error (No such device).		

SM_SIZE Default value: 30 SECTION 2

Size of the virtual memory for the *segmap* segment. The specification takes the form of the percentage of the real size of the main memory which is occupied by the *segmap* segment.

SYSTEM_UMASK Default value: 0 SECTION 2

Minimum value which the user may specify in the *umask*(1) command. This is a built-in shell command and is used to output or modify the current protection bit mask.

Parameters for input/output

DVHANGUP Default value: 0 SECTION 3

This parameter ends processes which are sending output to users who have logged off.

IDLE_MIGRATE Default value: 3 SECTION 3

The parameter serves to reduce the time required to find a free CPU.

MAXVDUNIT Default value: 1024 SECTION 2

Maximum number of virtual disks.

PPNUMLOCK Default value: 200 SECTION 3

Minimum number of physical memory pages which may be neither locked nor allocated. The system memory segments may be paged in and out. However, there are also segments which cannot be treated in this way and which must therefore remain present in the system. If fewer than *PPNUMLOCK* pages are available then no further pages which must remain in the system are allocated. This mechanism is intended to prevent deadlocks.

Network-related parameters

The network-related parameters are described in the “Network Administration” manual, chapter 11.

7.8.12 Computer-specific Parameters

<i>DEN_HWM_PAGES</i>	Default value: 2560	SECTION 4
<i>DEN_LWM_PAGES</i>	Default value: 2048	SECTION 4

These parameters are required by the DEN Ethernet controller.
DEN_HWM_PAGES specifies the maximum number of main memory pages which may be allocated to DEN. When this limit is reached, other processes must wait until memory is again available from the pool.
DEN_LWM_PAGES specifies the minimum amount of memory which DEN may use. Below this limit, memory is no longer freed. The kernel does not use either of these parameters.

<i>DPTGCHANS</i>	Default value: 10	SECTION 8
------------------	-------------------	-----------

Maximum number of virtual channels per physical V.24 line. The parameter only exists if the *SIcptg2* packet has been installed. For example, it defines the number of devices connected to a BA80: e.g. eight monitors and two printers.

<i>MAXNETUSERS</i>	Default value: <i>MAXUSERS</i>	SECTION 8
--------------------	--------------------------------	-----------

Maximum number of network users permitted in the system. This parameter is not directly used by the kernel. It is simply used for the computation of other kernel parameters (see figure 11 on page 173). If the size of the kernel is critical you can modify the value as follows: To save space you can reduce the value if the system executes no or only a few network processes. If, on the other hand, the system executes a large number of network processes, you will probably need to increase this value.

<i>MAXNFSCLIENTS</i>	Default value: 24	SECTION 8
----------------------	-------------------	-----------

Maximum number of RPC requests (RPC = Remote Procedure Call) which can be active simultaneously in the kernel. The parameter has no effect on user-level RPC calls. The default value is also the maximum value for this parameter. This is a hardware-defined kernel value which cannot be changed by modifying the value of the *stune* file. You may reduce the value if you make little or no use of NFS. However, the space saving is small.

MAXUSERS Default value: 512 SECTION 8

This parameter is not directly used by the kernel. It is simply used for the computation of other kernel parameters (see figure 11 on page 173). It does **not** limit the number of “login” processes.

QUEUED_LOCKS Default value: 0 SECTION 4

During program processing on multiprocessor systems, there are critical areas where only one processor may process certain data while the others have to wait until this first processor again becomes available. These areas are protected by “spinlocks”. For some time now it has also been possible to enter such data areas in a queue. If this parameter is set to 1 then only “queued locks” are used. If the parameter remains set to 0 then the system itself determines whether or not queued locks should be used. This parameter should not be modified.

SECURE_FS Default value: 0 SECTION 4

If this value is set to 1 then the “secure file system” is activated. This is only of use if the CSP add-on product is installed. This parameter should not be modified.

TAPE_NO_AUTOLOAD Default value: 0 SECTION 8

If this value is set to 0, the tape remains loaded for all Exabyte and Dat drives and is no longer, as was the case previously, unloaded if it is idle for several minutes.

8 Appendix

Below are some summaries to help you in your tuning activities.

8.1 A brief introduction to tuning

CPU

check	observe column	Measures
<i>sar -u</i>	113	Install additional CPU or optimize application
<i>sar -P</i>	83 Sum %usr + %sys high (~ 100%)	
<i>vmstat</i>	44	
<i>mpstat</i>	41 Sum us + sy high (~ 100%)	
<i>simon</i>	64 User+System CPU % high (~ 100%)	
<i>sar -q</i>	85 Load distribution to CPUs runq-sz high (see page 206)	Control processes or bind to a CPU
<i>ps -ef</i>	88 TIME high (several minutes)	If possible, reduce number of processes or move processes temporarily

Table 10: Brief introduction to tuning – resource CPU

Main memory

check	observe column	Measures
<i>sar -r</i>	91 freemem small (< 8 Mbyte)	Extend main memory
<i>vmstat</i>	93	Reduce number of memory-intensive processes
<i>mpstat</i>	41 fre small (< 8 Mbyte)	
<i>usr/ucb/ps -avx</i>	88 RSS high ⁽¹⁾	Optimize application
<i>sar -g</i>	99 ppgout/s and ppgin/s always high 101 (see page 207)	
<i>sar -p</i>		

Table 11: Brief introduction to tuning – resource main memory

1) Which values are considered “high” depends on the size of the main memory and the number of processes. In general, values of 1 Mbyte and higher are relevant.

Hard disks

check	observe column	Measures
sar -d	115 %busy	always > 60% ⁽¹⁾
	avque	(see page 207)
	r+w/s	(see page 207)
	avserv	(see page 207)
dkstat -p	39	I/O per partition and overall

Table 12: Brief introduction to tuning – resource hard disks

1) As a result of access optimization, the value at high loads may even be higher than 100% in the case of sequential read access (see sar(1)).

8.2 Maximum values for sar analysis

The following tables list critical maximum values for various *sar* outputs. These values are intended as guidelines. In certain cases, values which differ from those below may also be critical.

Computers with R4400 processor

sar	Column	Displayed information	CPU frequency (internal) in MHz			
			100	150	200	250
-u	113 %usr	Proportion of CPU time in user mode	Sum always ~ 100%			
	%sys	Proportion of CPU time in system mode	Sum always ~ 100%			
	int/s	Number of interrupts per second ⁽¹⁾	300	500	700	900
-w	103 pswch/s	Number of process switches per second	300	500	700	900
-q	84 runq-sz	Size of the run queue	20	30	40	50
-m	126 msg/s	Number of messages per second	300	400	500	650
	sema/s	Number of semaphore operations per second	300	400	500	650

Table 13: Maximum values for sar analysis – R4400 processor

sar	Column	Displayed information	CPU frequency (internal) in MHz			
			100	150	200	250
-g	98	ppgout/s	Number of pages paged out per second		always > 10 ⁽¹⁾	
-p	100	ppgin/s	Number of pages paged in per second		always > 10 ⁽¹⁾	
-d		vflt/s	Page faults during the translation of virtual to physical addresses		always > 100	
	114	%busy	Proportion of time during which a block device was busy performing a transmission job		always > 60%	
		avque	Average proportion of the time specified in %busy during which jobs were present in the job queue		always >> 1	
		r+w/s	Number of read/write operations per second		Formula for calculation see page 209	
		avserv	Average time during which data transmission jobs are processed		always > 100 ms	

Table 13: Maximum values for sar analysis – R4400 processor

Computers with R5000, R10000 or R12000 processors

sar	Column	Displayed information	CPU frequency (internal) in MHz						
			180 ⁽¹⁾	180 ⁽²⁾	200 ⁽³⁾	250 ⁽³⁾	285 ⁽³⁾		
-u	113	%usr	Proportion of CPU time in user mode		Sum always ~ 100%				
		%sys	Proportion of CPU time in system mode		Sum always ~ 100%				
		int/s	Number of interrupts per second ⁽¹⁾		1600	3000	3500	4500	5200
-w	103	pswch/s	Number of process switches per second		800	1300	1500	1900	2200
-q	84	runq-sz	Size of the run queue		40	50	50	60	70
-m	126	msg/s	Number of messages per second		500	500	600	750	900
		sema/s	Number of semaphore operations per second		500	500	600	750	900

Table 14: Maximum values for sar analysis – R5000, R10000 or R12000 processor

sar	Column	Displayed information	CPU frequency (internal) in MHz				
			180 ⁽¹⁾	180 ⁽²⁾	200 ⁽³⁾	250 ⁽³⁾	285 ⁽³⁾
-g	98	ppgout/s	Number of pages paged out per second				
			always > 10 ⁽¹⁾				
-p	100	ppgin/s	Number of pages paged in per second				
			always > 10 ⁽¹⁾				
		vflt/s	Page faults during the translation of virtual to physical addresses				
			always > 100				
-d	114	%busy	Proportion of time during which a block device was busy performing a transmission job				
			always > 60%				
		avque	Average proportion of the time specified in %busy during which jobs were present in the job queue				
			always >> 1				
		r+w/s	Number of read/write operations per second				
			Formula for calculation see page 209				
		avserv	Average time during which data transmission jobs are processed				
			always > 100 ms				

Table 14: Maximum values for sar analysis – R5000, R10000 or R12000 processor

- 1) R5000
- 2) R1000
- 3) R12000

Notes to table 13 and table 14

CPU frequency (internal)
That is the frequency at which the CPU operates internally. This frequency is also referred to as the internal clock speed. It is displayed by the *mpcntl -I* command (CLOCK column, see page 82). This should not be confused with the external frequency or clock speed which is much less.

int/s
The number of interrupts per second is also measured by the commands *mpstat(1M)* (see page 40) and *vmstat(1M)* (see page 42) and is output in the column *int*.

ppgout/s
ppgin/s
Ideally these values should be zero.

r+w/s
The maximum number of read/write operations per second which is possible for a disk type can be calculated using the formula below:

sequential:

speed of rotation

60

*

95

100

random:

speed of rotation

60

*

95

100

*

0,7

For example: MP81 hard disk at 5400 rpm

sequential:

~ 85 I/Os

random:

~ 60 I/Os

8.3 Technical data for hard disks

Hard disk	MP12	MP14	MP25	MP36	MP38
Connector type	50Pin	50Pin	68Pin	50Pin	80Pin (SCA-2)
SCSI interfaces	SE, 8 Bit	SE, 8 Bit	DF, 8 Bit	SE, 8 Bit	SE, 16 Bit SE, 8 Bit
Capacity, formatted (GB)	4,29	2,147	4,29	2,276	2,276
Sector size (Byte)	512	512	512	512	512
No. of cylinders	3711	3510	3711	5952	5952
No. of heads	21	11	21	6	6
Cache size, net (KB)	924	480	924	512	512
Rotation speed (rpm)	7200	7200	7200	7200	7200
Seek time, track to track (ms)	1,6/1,9 ⁽¹⁾	1,6/1,9 ⁽¹⁾	1,6/1,9 ⁽¹⁾	0,9	0,9
Seek time, average (ms)	9,0/10,0 ⁽¹⁾	9,0/10,0 ⁽¹⁾	9,0/10,0 ⁽¹⁾	8,0/8,3 ⁽¹⁾	8,0/8,3 ⁽¹⁾
Seek time, max. (ms)	18,0/20,0 ⁽¹⁾	18,0/20,0 ⁽¹⁾	18,0/20,0 ⁽¹⁾	18,0	18,0
Latency, average/max. (ms)	4,17/8,3	4,17/8,3	4,17/8,3	4,17/8,3	4,17/8,3
Transfer rate, extern (MB/s)	10	10	20	10	20/10 ⁽²⁾

Table 15: Technical data for hard disks – Part 1

¹⁾ Read / Write

²⁾ 16 Bit / 8 Bit

Hard disk	MP46	MP47	MP48	MP81	MP85
Connector types	50Pin	68Pin	80Pin (SCA-2)	50Pin	50Pin
SCSI interfaces	SE, 8 Bit	DF, 16 Bit DF, 8 Bit	SE, 16 Bit SE, 8 Bit	SE, 8 Bit	SE, 8 Bit
Capacity, formatted (GB)	4,552	4,552	4,552	1,05	2,147
Sector size (Byte)	512	512	512	512	512
No. of cylinders	5952	5952	5952	2874	2621
No. of heads	10	10	10	8	19
Cache size, net (KB)	512	512	512	512	512
Rotation speed (rpm)	7200	7200	7200	5400	5400
Seek time, track to track (ms)	0,9	0,9	0,9	1,8	1,8
Seek time, average (ms)	8,0/8,3 ⁽¹⁾	8,0/8,3 ⁽¹⁾	8,0/8,3 ⁽¹⁾	9,5/11,0 ⁽¹⁾	9,5/11,0 ⁽¹⁾
Seek time, max. (ms)	18	18	18	19,0	19,0
Latency, average/max. (ms)	4,17/8,3	4,17/8,3	4,17/8,3	5,6/11,11	5,6/11,11
Transfer rate, extern (MB/s)	10	20/10 ⁽²⁾	20/10 ⁽²⁾	10	10

Table 16: Technical data for hard disks – Part 2

¹⁾ Read / Write²⁾ 16 Bit / 8 Bit

Hard disk	MP94	MP96	MP97	MP98
Connector types	80Pin (SCA-2) 50Pin	50Pin	68Pin	80Pin (SCA-2)
SCSI interfaces	SE, 16 Bit SE, 8 Bit	SE, 8 Bit	DF, 16 Bit DF, 8 Bit	SE, 16 Bit SE, 8 Bit
Capacity, formatted (GB)	9,105	9,104	9,104	9,104
Sector size (Byte)	512	512	512	512
No. of cylinders	7501	5952	5952	5952
No. of heads	10	20	20	20
Cache size, net (KB)	924	1024	1024	1024
Rotation speed (rpm)	7200	7200	7200	7200
Seek time, track to track (ms)	1,1/1,4 ⁽¹⁾	0,9	0,9	0,9
Seek time, average (ms)	7,4/8,2 ⁽¹⁾	7,5/8,5 ⁽¹⁾	7,5/8,5 ⁽¹⁾	7,5/8,5 ⁽¹⁾
Seek time, max. (ms)	16,0/18,0 ⁽¹⁾	18,0	18,0	18,0
Latency, average/max. (ms)	4,17/8,3	4,17/8,3	4,17/8,3	4,17/8,3
Transfer rate, extern (MB/s)	20/10 ⁽²⁾	10	20/10 ⁽²⁾	20/10 ⁽²⁾

Table 17: Technical data for hard disks – Part 3

1) Read / Write

2) 16 Bit / 8 Bit

Hard disk	HD02	HD04	HD05	HD07
Connector types	80Pin (SCA-2)	80Pin (SCA-2)	80Pin (SCA-2)	80Pin (SCA-2)
SCSI interfaces	SE, 16 Bit	LVD SE, 16 Bit	SE, 16 Bit	LVD SE, 16 Bit
Capacity, formatted (GB)	4,55	9,1	9,105	18,21
Sector size (Byte)	512	512	512	512
No. of cylinders	6526	6962	6526	6476
No. of heads	8	12	16	24
Cache size, net (KB)	442	840	480	840
Rotation speed (rpm)	10000	10000	10000	10000
Seek time, track to track (ms)	0,98/1,24 ⁽¹⁾	0,8/1,5 ⁽¹⁾	0,8/1,2 ⁽¹⁾	0,8/1,5 ⁽¹⁾
Seek time, average (ms)	7,7/8,7 ⁽¹⁾	5,4/6,2 ⁽¹⁾	8,2/9,7 ⁽¹⁾	5,4/6,2 ⁽¹⁾
Seek time, max. (ms)	18,2/19,2 ⁽¹⁾	12,2/13,2 ⁽¹⁾	19,2/20,2 ⁽¹⁾	12,2/13,2 ⁽¹⁾
Latency, average/max. (ms)	2,99/5,98	2,99/5,98	2,99/5,98	2,99/5,98
Transfer rate, extern (MB/s)	40	80/40 ⁽²⁾	40	80/40 ⁽²⁾

Table 18: Technical data for hard disks – Part 4

¹⁾ Read / Write²⁾ LVD / 16 Bit

Hard disk	HD11	HD14	HD17
Connector types	80Pin (SCA-2)	80Pin (SCA-2)	80Pin (SCA-2)
SCSI interfaces	LVD SE, 16 Bit	LVD SE, 16 Bit	LVD SE, 16 Bit
Capacity, formatted (GB)	4,551	9,105	18,2
Sector size (Byte)	512	512	512
No. of cylinders	7501	7501	7501
No. of heads	5	10	20
Cache size, net (KB)	924	924	924
Rotation speed (rpm)	7200	7200	7200
Seek time, track to track (ms)	1,1/1,4 ⁽¹⁾	1,1/1,4 ⁽¹⁾	1,1/1,4 ⁽¹⁾
Seek time, average (ms)	5,4/6,2 ⁽¹⁾	7,4/8,2 ⁽¹⁾	7,9/8,7 ⁽¹⁾
Seek time, max. (ms)	16,0/18,0 ⁽¹⁾	16,0/18,0 ⁽¹⁾	16,0/18,0 ⁽¹⁾
Latency, average/max. (ms)	4,17/8,3	4,17/8,3	4,17/8,3
Transfer rate, extern (MB/s)	80/40 ⁽²⁾	80/40 ⁽²⁾	80/40 ⁽²⁾

Table 19: Technical data for hard disks – Part 5

¹⁾ Read / Write

²⁾ LVD / 16 Bit

Hard disk	HD24 ⁽¹⁾	HD28 ⁽¹⁾	HD38 ⁽¹⁾
Connector types	80Pin (SCA-2)	80Pin (SCA-2)	80Pin (SCA-2)
SCSI interfaces	LVD SE, 16 Bit	LVD SE, 16 Bit	LVD SE, 16 Bit
Capacity, formatted (GB)	9,1	18,2	36,4
Sector size (Byte)	512	512	512
No. of cylinders	9866	9866	10200
No. of heads	5	10	19
Cache size, net (KB)	2048	2048	2048
Rotation speed (rpm)	10000	10000	10000
Seek time, track to track (ms)	no specification possible at present		
Seek time, average (ms)	7,5/8,5 ⁽²⁾	7,5/8,5 ⁽²⁾	7,5/8,5 ⁽²⁾
Seek time, max. (ms)	16,0/17,0 ⁽²⁾	16,0/17,0 ⁽²⁾	16,0/17,0 ⁽²⁾
Latency, average/max. (ms)	2,99/5,98	2,99/5,98	2,99/5,98
Transfer rate, extern (MB/s)	80/40 ⁽³⁾	80/40 ⁽³⁾	80/40 ⁽³⁾

Table 20: Technical data for hard disks – Part 6

¹⁾ Planned after Reliant UNIX 5.45A10

²⁾ Read / Write

³⁾ LVD / 16 Bit

Hard disk	HD06	HD09	HD19 ⁽¹⁾	HD29 ⁽¹⁾	HD39 ⁽¹⁾
Connector type	40Pin (SCA-2)				
Interface	Fibre Channel				
Capacity, formatted (GB)	9,1	18,2	9,1	18,2	36,4
Sector size (Byte)	512	512	512	512	512
No. of cylinders	6962	6962	9866	9866	10200
No. of heads	12	24	5	10	19
Cache size, net (KB)	967,5	967,5	2048	2048	2048
Rotation speed (rpm)	10000	10000	10000	10000	10000
Seek time, track to track (ms)	0,8/1,1 ⁽²⁾	0,8/1,1 ⁽²⁾	no specification possible at present		
Seek time, average (ms)	5,4/6,2 ⁽²⁾	5,9/6,7 ⁽²⁾	7,5/8,5 ⁽²⁾	7,5/8,5 ⁽²⁾	7,5/8,5 ⁽²⁾
Seek time, max. (ms)	12,2/13,2 ⁽²⁾	18,0/20,0 ⁽²⁾	16,0/17,0 ⁽²⁾	16,0/17,0 ⁽²⁾	16,0/17,0 ⁽²⁾
Latency, average/max. (ms)	2,99/5,98	2,99/5,98	2,99/5,98	2,99/5,98	2,99/5,98
Transfer rate, extern (MB/s)	80	80	80	80	80

Table 21: Technical data for hard disks – Part 7

¹⁾ Planned after Reliant UNIX 5.45A10

²⁾ Read / Write

Glossary

address

An address refers to the location of data in the computer's physical or virtual memory.

allocation unit

This is a self-contained unit in a VxFS file system for administering the inodes and blocks. The allocation unit corresponds to cylinder group in the UFS file system.

ASCII file

See text file.

big page

A big page is a large contiguous memory area. The use of big pages reduces the number of dynamically required entries in the TLB (Translation Lookaside Buffer). The following types of big page exist in Reliant UNIX 5.44:

- for shared memory data (16 Mbytes)
- for user text (4 – 32 Mbytes)
- for page tables (64 Kbytes, 256 Kbytes oder 1 Mbyte)

There is only a single big page in the system.

block device

This is a device on which a file system can be mounted. Typical block devices are tape or disk devices for the permanent storage of data. The designation “block device” comes from the fact that data is transferred to such devices in blocks. See also character device.

Board Debug Monitor

The Board Debug Monitor (BDM) is a monitoring program on a special board:

RM600-xxx: CSI¹ board
RM600 E20/E60: HIOS² master board
RM600 E30/E70: EHIOS³ master board

¹ CSI = Central Services & Interfaces

² HIOS = High Performance Input Output System

³ EHIOS = Enhanced High Performance Input Output System

It starts to operate automatically when the Master Test Handler has correctly completed its tests. Depending on the boot mode which has been set it either automatically starts Boot1 or waits for user input. The BDM issues the following prompt:

RM600-xxx: BDM(MBII 0 t)>

RM600 E: BDM(SPB 8 t)>

boot

Bootting is the term used to describe starting the operating system. First of all the kernel is loaded from a data store (for example, the system disk) into main memory. Bootting involves a number of boot phases.

RM200, RM300, RM400: Prom Monitor and SASH

RM600-xxx, RM600 E: Board Debug Monitor, Boot1 and Boot2

No login or process remains valid after a reboot.

Boot1

RM600-xxx, RM600 E:

Boot1 (First Level Boot) loads Boot2 from the load device (system disk or CD-ROM) into the main memory and starts it.

Boot2

RM600-xxx, RM600 E:

Boot2 (Second Level Boot) is an independent operating system. It is located on the system disk in partition 10. It is loaded into main memory by Boot1 where it is executed as a bootstrap process. Depending on the boot mode which has been set, it either automatically loads the Reliant UNIX operating system or issues the prompt `boot2:` and waits for user input. Sometimes the term SASH (Stand-Alone SHell) is used as a synonym for Boot2.

boot mode

There are a number of modes in which the system can be booted:

- Autoboot and manual boot

In autoboot, all boot phases are run through automatically until the login prompt of the Reliant UNIX operating system appears. With manual booting, the boot procedure is stopped in the following boot phases:

RM200, RM300, RM400: Prom Monitor and SASH

RM600-xxx, RM600 E: Board Debug Monitor, Boot1 and Boot2

A prompt appears and the user can enter commands.

- Logical and physical boot

The name is derived from whether the boot medium is accessed using the logical or the physical address. The address is stored in the boot string. The physical boot mode is required if the operating system is not to be booted from the active system disk, but instead from another storage medium (e.g. from the CD-ROM „Reliant UNIX“). The logical boot mode offers the advantage that a number of system disks that can be connected to any controller in the system cabinet can be administered. The logical addresses are assigned to the physical addresses here via a root partition list.

Prerequisites for the RM600-xxx:

The physical boot mode is only possible if the system disk is connected to the CSI. The logical boot mode is only possible with Reliant UNIX Version 5.43 or higher and the firmware version V2.42 for the CSI.

break

A break is

1. a signal which is generated by “break” or by a break character and which usually causes a process to terminate,
2. a signal generated by the hardware or by a peripheral device,
3. in general terms: any signal.

buffer

A buffer is an area used for the intermediate storage for variable length input/output data. Such data is subdivided into smaller units which are more easily usable for system functions. Thus buffers are used by the file system [3] as well as by the standard input/output functions (stdio).

buffer pool

A buffer pool is a memory area which is available to the file system [3] for the intermediate storage of blocks. Unlike in the case of character devices, the input/output for block devices is always performed via the buffer pool. In this way read/write operations can be performed independently of the size of the device blocks.

character device

Examples of character devices are terminals or the null device. No file systems [1] can be mounted on such devices. See also block device.

child process

See fork.

command

A command is

1. an instruction issued to the shell. This normally starts a program [1] as a child process (see fork).
2. in its wider sense: any executable file.

configuration

The configuration of a computer refers to

1. the characteristics of the software which is running on the computer. The characteristics of the operating system are influenced by: certain parameters (e.g. `$SHELL`, `$TZ`, `$PS1`), certain files (e.g. `/etc/profile`, `/etc/passwd`), the tunable parameter (defined in the files `/etc/conf/cf.d/mtune` and `/etc/conf/cf.d/stune`).
2. the hardware components belonging to the computer. This includes controllers, hard disks and drives as well as the connected peripheral devices.

configuring the kernel

This refers to

1. modifications to the tunable parameters designed to enhance the performance of the operating system (tuning).
2. generation of the kernel in order to integrate modifications to the tunable parameters in the executable version of the kernel.

console

The term console refers to the terminal which is directly connected to the computer. The system administrator uses this terminal to communicate with the computer.

controller

A controller is an intelligent control unit that controls more or less independently data input between the central processor and a peripheral device. The controller generally consists of a slot card that is inserted in a slot on a system board and is connected with the peripheral devices via a cable.

core image

A core image (or memory image) is a copy of all the segments of a process. Such a copy is created if the process is not terminated “normally” (e.g. by a signal). The copy may be located in main memory, in the swap area or in a core image file named *core*. The *core* file is used for troubleshooting. It is always created in the current directory of the process in question.

crash

Hardware or software errors can create conditions which the operating system is unable to cope with. It is then automatically terminated and “crashes”. Such conditions occur, for example, if

- the operating system is unable to allocate any more resources,
- it can no longer manage processes,
- it can no longer operate system calls,
- the power supply is no longer stable.

cron

The *cron* command generates a daemon process which starts programs at a specified time.

cylinder

This refers to the total of all the tracks on a hard disk which are located at the same distance from the disk's center of rotation.

daemon

The term daemon is used to describe a process which runs in the background (in most cases permanently) and performs a public task for the entire system. For example: *calendar*(1) and *cron*(8).

demand paging

This is a technique used by the kernel to start processes. The kernel provides the virtual address space for the process but does nothing to load the process into memory. It then jumps to the start address of the process. As this is not present in memory, a page fault is triggered and the paging mechanism loads the first page of the process from the hard disk. See also paging.

device

This term is used to designate

1. a file which is neither a regular file nor a directory, for example a tape device, the null device or a device unit.
2. a physical input/output unit.

directory

A directory is a file which contains a catalog of filenames [2]. As the organizing principle underlying file systems [2], a directory consists of entries which designate files (and therefore also other directories). A directory is therefore a node in the directory tree.

directory entry

A directory entry is

1. the assignment of a name to an inode number, i-number. Each entry in a directory has this form.
2. the name part of such an assignment.

directory tree

The directory tree is the entire tree consisting of each available directory. In this tree, each directory can be reached via a series of subdirectories starting from root directory.

dump

See core image.

environment

An environment is

1. a set of character strings which are made available to a process together with its arguments when it executes a file [2]. Normally the environment is inherited following `exec(2)` operations,
2. a particular environment [1] which is managed by the shell.

error

An error is said to have occurred when, for hardware or software reasons, a system or user process cannot be executed successfully. In such a case, the operating system may send an error message to the console or to the user terminal.

exec

exec is a system call which allows processes to execute programs.

executable file

An executable file is

1. an object file which can be copied to the address space of a process where it is run as the code of this process.
2. a file whose execute permissions have been set. This is either an executable file [1] or a shell script.

execute

This term has two meanings:

1. general: to run a program
2. the overlaying of a process's text and data segments by a program [1].

file

A file is

1. general: a possible source for an input or a destination for an output,
2. special: an inode and/or the associated file contents, i. e. a regular file, a special file or a directory,
3. a directory entry; different directory entries may refer to the same file [2],
4. in general terms: a regular file.

If the execute permissions are set (see permissions) then the file is set to be executable (see executable file). If the file is used as a source for input or as a destination for output then the file is said to be open (see open file). Multiple instances of a file may be open simultaneously.

filename

A filename is

1. a *pathname*
2. the final component of a *pathname*

file system

A file system is

1. a collection of files which can be mounted on a block special file. The inode associated with each file appears precisely once in the file system's i-list. It is possible to access each file via a *path* from the root directory of the file system.
2. the totality of all the files [1] on the computer.
3. that part of the kernel which is responsible for file systems [1].

flush

This function is used to empty a buffer in order, for example, to remove unwanted input/output data when a break occurs or to force the output of the buffered data when standard I/O functions are used.

fork

This system call is used to duplicate a process with the result that there are subsequently a parent and a child process which possess separate (although initially identical) text segments, data segments and stack segments.

group

This term designates

1. a set of permissions which, together with the owner permissions, regulate file access.
2. a set of userids which can assume the priorities of a group [1].
3. the groupid of a file.

groupid

The groupid is an integer value which is usually assigned to one or more login names. If a process generates files then these, as owners, are assigned the userid of the process and, as a group [3], its groupid.

hard disk

hard disks are data media which are normally installed directly in the computer itself. This differentiates them from removable data media such as diskettes, cartridges and CDs. The system disk is of particular importance since it is here that the operating system is stored. Each hard disk is subdivided into a number of areas which are referred to as partitions. A list of the available partitions is stored in a partition table.

i-list

The list contains all the inodes belonging to a file system [1]. The reference to this list is performed via the inode number, i-number.

inode

The term inode (from information node) is used to describe an individual data structure in the i-list. There is precisely one inode corresponding to each file [2] of a file system [1]. This contains information which is required for the memory of this file. The reference to the inode table is provided by the inode number, i-number. An inode contains the following information: file type, permissions, number of links, owner, userid, group, groupid, size in bytes, date of the last inode modification, date of the last modification, date of the last access, references to file contents.

inode number, i-number

A number is assigned to each inode. In combination with the device number, the inode number provides an unambiguous reference to the associated i-list in a file system [1].

intent log

This is a reserved memory area at the start of the VxFS file system on the hard disk in which all not yet completed changes to the file system structure are logged. After a system error, utility *fsck*(1M) can be used to very quickly re-establish the file system integrity using this intent log.

kernel

The kernel is the actual operating system. It is located on the system disk in the file */stand/unix*. The file is approximately 5 Mbytes in size. During the boot operation the file is loaded into main memory where it remains memory-resident. You can use the command */etc/conf/bin/idbuild* to generate a new kernel. This is then automatically loaded the next time the system is booted.

kernel address space

This address space is a part of main memory containing kernel data and code which can only be accessed by the kernel.

link

A link is

1. an additional directory entry for an existing file. Such links can be created using the command *link(1M)* and deleted with *unlink(1M)*,
2. in its extended sense: a normal directory entry,
3. in general terms: any directory entry with the exception of the presumed main entry of a predefined inode which cannot be referenced using either normal links [1] or symbolic links.

login

A login is:

1. the program which enables a user to log onto the operating system.
2. the procedure for logging onto the operating system.
3. in the wider sense: the computer session which commences after login [2].

Master Test Handler

RM600-xxx, RM600 E:

The Master Test Handler (MTH) is automatically started when the computer is powered on. It checks the hardware and logs the results to the console. This log contains a test status for each tested module. If an error is detected in a module then the MTH prevents operating system startup.

memory

The term memory is used with the following meanings:

1. as a synonym for core image,
2. as the physical memory which represents the available space in main memory.
3. as a main memory extension. Thanks to virtual memory management, programs can then use storage space on the hard disk as a main memory extension.

memory disk

A memory disk is a main memory area which makes the same interfaces as are provided by a hard disk available to users and application programs via a corresponding device driver. Memory disks can also be used as sections of mirror disks. This combines the security of mirror disks with the read speed of a memory disk.

memory image

See *core image*.

message queue

Processes can exchange messages with one another. These messages are entered in a queue from which they can be fetched by the processes. This mechanism is known as interprocess communication (IPC). IPC also includes semaphores and shared memory.

mirror disk

Mirror disks are virtual disks which perform all input/output operations on two disks simultaneously. The various components of a mirror disk may themselves be physical or virtual disks. Mirror disks considerably improve the security and availability of hard disks and consequently of the entire system.

mount

Mounting a file system extends the directory hierarchy. This occurs when the root directory of the new file system [1] is assigned to a directory entry in an already mounted file system. This task is performed using the *mount(1M)* command, while the *umount(1M)* command performs the reverse.

null device

The device file special file (*/dev/null*) always provides end-of-files on read accesses. In the event of write accesses it discards all the data received.

open file

An open file is

1. the source or destination for input or output.
When you open a file or generate a pipe you receive a file descriptor for the open file. Following a fork call, open files are available for shared use are again available following an exec call.
2. in general terms: a file which has been opened.

However, an open file need not necessarily exist in a file system [1]. Multiple instances of a file may also be open simultaneously.

operating system

The operating system is a program for the management of computer resources. It is responsible, for example, for input/output procedures and for process planning. This frees user programs from the need to perform these tasks. Well-known operating systems are DOS, UNIX and Reliant UNIX.

other

The term other is used with two meanings:

1. to designate the access permissions to a file file for all processes whose userid differs from that of the owner of the file and whose groupid differs from that of the group to which the file belongs.
2. as the normal designation of the default group [2] which is allocated at login.

owner

The owner of a file is the userid of the process which generated the file. The owner has special permissions for its file.

page

The page is the fundamental unit used in virtual memory management. The default page size is 16 Kbytes (see *pagesize(1)*).

page fault

When the virtual memory management accessess a virtual address but is unable to find the corresponding page in main memory, a page fault occurs. The required page must then be fetched. This procedure is also used for demand paging.

paging

This refers to the process of dividing programs into pages which can then be transferred from the main memory to the hard disk by the virtual memory management or paging daemon (or transferred back from the hard disk).

parent process

See fork.

partition

Each hard disk is subdivided into a number of areas (partitions) which can overlap. The existing division (partitioning) is stored in a partition table on the hard disk. The partitions of the system disk must have a certain minimum size. The SASH, which is required to boot the computer, is stored in partition 10 of the system disk.

partition table

The partitioning of each hard disk is defined in a partition table. This table is stored on the hard disk. Partition tables are created and managed using the *dkpart*(8) command.

path, pathname

These two terms are synonymous and refer to the name string which designates a file. A relative pathname always starts from the current directory. If, for example, the current directory contains the subdirectory *A* which contains the subdirectory *B* which in its turn contains the file *xyz*, then this file is designated *A/B/xyz*. A complete pathname always begins at the root file system (*/*), e.g. */home/krenz/A/B/xyz*.

performance

The performance of the global system (hardware, operating system, application) is influenced by a large number of factors: the configuration of the input/output devices, the installed CPU, the number of users, the operating mode, the software configuration. The performance is expressed in terms of throughput and response times. tuning measures are intended to remove bottlenecks in order to improve performance.

permissions

Permissions regulate certain file access modes: read access, write access and execute permissions (in the case of a directory this is the search permission). Permissions are distributed separately to owners, groups and others.

physical memory

See memory.

pipe

A pipe is a direct connection for the flow of data between processes. The output data from one process become the input data to the other process.

process

A process is a connected series of statements. When a process is started three files are usually opened: the standard input file (*stdin*), the standard output file (*stdout*) and the standard error output file (*stderr*) from the default library *stdio*. Initially all three files refer to the terminal.

The properties of a process are defined by its core image. The core image contains the command counter and information about the current directory, the set of opened files, the monitoring terminal and the *userid* and *groupid*.

process id, process number

The process id is an integer value used to identify a process.

profile

A profile is

1. an optional shell script named *.profile*. The shell usually works through this script when a user logs on in order to set up the environment [3] and other working conditions which the user is used to.
2. a file used to record the time-related profile of a process. It contains regularly polled command counter values.

program

A program is

1. an executable file
2. a process

Prom Monitor

RM200, RM300, RM400:

The Prom Monitor is a firmware prom component. It starts to operate as soon as the power-up phase and basic initialization have been terminated correctly. The Prom Monitor checks the hardware and outputs the result at the screen. Depending on the boot mode which has been set, it either automatically starts the SASH or issues the prompt `>>` and waits for user input.

queue

A queue is a list of actions which are waiting for processing.

raw device

A raw device is a block device which is operated unbuffered for read and write operations. During such operations, the data records are adapted to the device-specific block size of the physical device.

response time

Time difference between input and output at a terminal. The response time depends on the current throughput. The aim of tuning measures is to reduce the response time.

root

root is

1. the identification of the *root* user (login name). As system administrator or superuser, this user possesses special permissions (*root* permissions).
2. a special directory (/) which represents the starting point (root directory) of the directory hierarchy in a file system [1]. This directory is also the login directory of the *root* user.

root directory

A root directory is

1. the starting point for the entire file system [2]. It has the pathname /.
2. the starting point of a directory tree in a file system [1].

SASH

RM200, RM300, RM400:

The SASH (Stand-Alone-SHell) is an independent operating system. The SASH is located in partition 10 of the system disk. It is loaded into main memory by the Prom Monitor where it is executed as a bootstrap procedure. Depending on the boot mode which has been set, it either automatically loads the UNIX operating system or issues the prompt *sash*: and waits for user input.

scheduler

The scheduler is a functional complex within the kernel. It is responsible for assigning computation time to processes.

search path

The search path of the shell consists of a list of directory names which determine the meaning of a command. The command is prefixed with the directory names one by one until this enables an executable file to be found. The search path is specified in the shell variable *\$PATH*.

semaphore

Semaphores are status variables. They are used to synchronize access to critical resources and may assume a number of different values.

Processes can query the value of semaphores or wait until the semaphore assumes a particular value. Semaphores are a component of inter-process communication (IPC). This also includes message queues and shared memory.

set userid bit

The set userid bit represents a special access permission for an executable file. It results in the permissions of the owner of a file being granted to the executing process. In contrast to the real userid, the process is assigned the userid of the owner of the executable file as an effective userid when it is started.

shared memory

This is a component of interprocess communication (IPC) which allows multiple processes to share the same data segment. Interprocess communication also includes message queues and semaphores.

shell

The shell is

1. the program *sh*(1) which causes other programs to run in response to the input of commands. Usually the shell is the program set up by the system administrator which is started when a user logs onto the operating system (see also login).
2. by analogy, the designation of the program which is started following logon.

shell procedure

A synonym for shell script.

shell script

A shell script is an executable file whose contents serve as command input for the shell.

signal

A signal is an exception condition which causes a process to be automatically ended or to diverge from normal flow-control. See also break and trap.

special file

A special file consists of an inode which points to a device.

startup

See boot.

superuser

See system administrator.

swap area

The swap area is that part of the hard disk which receives the so-called “anonymous” memory contents during swapping or paging operations. It is an independent area which does not form part of the file system.

swapping

In a swapping operation, all the pages of a process are transferred from main memory to the hard disk in order to make space for other processes. In addition, the process is identified as “not loaded”.

symbolic link

A symbolic link is a file whose content is the *pathname* of another file. When a symbolic link is accessed, the access is directed to the file with the specified *pathname*.

SYSADM

SYSADM is an interactive program package for system administration. It is started by the command *sysadm(1M)* and supports most system administration tasks within its own user environment.

system administrator

The system administrator or superuser is the user who is logged on with the login name *root* and the *userid* 0. Only the system administrator is authorized to execute certain administration tasks. The rights required for this purpose are also called *root* authorization. He can access any file and may activate certain system calls.

system call

The term system call is used to designate

1. a set of operating system functions. These functions assign, start, monitor, handle and terminate all operating system operations.
2. calls sent by user processes in order to execute system-dependent functions. These include input/output functions and process generation.

system disk

A computer may contain multiple hard disks. The system disk is the hard disk which contains the *root* file system (/). The partitions of the system disk must be of a certain minimum size. Partition 10 of the system disk is used to store the SASH which is needed to boot the computer.

text file

The contents of a text or ASCII file consists solely of ASCII characters.

throughput

Volume of data (in bytes) which is transferred per second. The specification always refers to a physical component of the computer, i. e. to the data rate from/to the CPU, from/to main memory, from/to the hard disk. The higher the throughput is, the lower the response time. The aim of tuning measures is to increase throughput.

trap

The term trap refers to a software mechanism used to detect and interpret certain hardware and software conditions. Traps are used to intercept signals (or breaks) and take the corresponding measures.

tunable parameter

Tunable parameters are used to define the sizes and limit values for various control structures of the Reliant UNIX operating system. In this way it is possible to a certain extent to adapt the operating system to particular operating conditions. These parameters are defined in the files */etc/conf/cf.d/mtune* and */etc/conf/cf.d/stune*.

tuning

Tuning refers to all the measures which are taken in order to improve the performance of the system as a whole (hardware, operating system, application).

userid

The userid is an integer value which is normally allocated to a login name. The userid of a process is inherited by all the processes which are generated by this process (via fork). It is also entered as the owner of all the files which are created by the process.

virtual disk

Virtual disks are formed by combining the partitions (slices) of multiple physical disks to form a logical partition. The memory disk represents a special type of virtual disk and can be used fully or only partly as a virtual disk.

virtual memory

A memory management procedure which is used by most modern operating systems. See also memory.

Abbreviations

ARP	Address Resolution Protocol
ASCII	American Standard Code for Information Interchange
BDM	Board Debug Monitor
ccNUMA	cache coherent Non Uniform Memory Access
CD-ROM	Compact Disk - Read Only Memory
COFF	Common Object File Format
CPU	Central Processing Unit
CSI	Central Services & Interfaces
DCS	Disk Controller SCSI
DF	Differential ended (SCSI)
DNLC	Directory Name Lookup Cache
EHIOS	Enhanced High Performance Input Output System
FDDI	Fiber Distributed Data Interface
FIPS/BSD	Federal Information Processing Standards / Berkeley Software Department
FPU	Floating Point Processor Unit
GMT	Greenwich Mean Time
HIOS	High Performance Input Output System
I/O	Input/Output

Abbreviations

ICMP	Internet Control Message Protocol
ID	Identifier
IP	Internet Protocol
IPC	Interprocess Communication
ISO	International Standardization Organization
KMA	Kernel Memory Allocation
LAN	Local Area Network
LVD	Low Voltage Differential (SCSI)
MHz	Megahertz
MST	Middle European Summertime
MTH	Master Test Handler
MTU	Maximum Transmission Unit
NFS	Network File System
NUMA	Non Uniform Memory Access
PID	Process ID
ptem	Pseudo Terminal Emulation Module
RAID	Redundant Array of Independent Disks
RAM	Random Access Memory
RFS	Remote File System

RPC	Remote Procedure Call
RSS	Resident Set Size
SAD	STREAMS Administration Driver
sar	System Activity Reporter
SASH	Stand-Alone Shell
SCA	Single Connection Attachment
SCSI	Small Computer System Interface
SE	Single Ended (SCSI)
SLC	Second Level Cache
SLIP	Serial Line Internet Protocol
SPbus	Synchronous Pipelined Bus
TCP	Transmission Control Protocol
TLB	Translation Lookaside Buffer
UDP	User Datagram Protocol
UFS	UNIX File System
UUCP	UNIX to UNIX Copy
VM	Virtual Memory
VxFS	VERITAS File System
WAN	Wide Area Network

Figures

Figure 1:	Characteristics of the various storage media	6
Figure 2:	Loading a program	8
Figure 3:	Areas in dynamically managed virtual main memory . . .	10
Figure 4:	ccNUMA architecture in the RM600 E	21
Figure 5:	Sequence of sar evaluations	33
Figure 6:	Menu SYSADM – performance – sar	34
Figure 7:	Displaying the most important system activities using SYSADM 36	
Figure 8:	Displaying the system load with xload	56
Figure 9:	Main window of SINIX/monitor	60
Figure 10:	Structure of a virtual disk of disk type striped	140
Figure 11:	The interdependence of kernel parameters	173
Figure 12:	RM200, RM300, RM400: Modifying the kernel parameters using SYSADM	175
Figure 13:	RM200, RM300, RM400: Generating a new kernel using SYSADM	178

Tables

Table 1: Kernel parameters which specify the size of system tables	26
Table 2: Options of the sar(1) command	32
Table 3: sar options in the SYSADM menu item System Activity Report	35
Table 4: Commands corresponding to the SINIX/monitor display	61
Table 5: Commands for load measurement	69
Table 6: Overview of proposed shell procedures	70
Table 7: The most important kernel parameters for tuning operations	165
Table 8: Modifiable kernel parameters, part 1	166
Table 9: Modifiable kernel parameters, part 2	167
Table 10: Brief introduction to tuning – resource CPU	205
Table 11: Brief introduction to tuning – resource main memory	205
Table 12: Brief introduction to tuning – resource hard disks	206
Table 13: Maximum values for sar analysis – R4400 processor	206
Table 14: Maximum values for sar analysis – R5000, R10000 or R12000 processor	207
Table 15: Technical data for hard disks – Part 1	210
Table 16: Technical data for hard disks – Part 2	211
Table 17: Technical data for hard disks – Part 3	212
Table 18: Technical data for hard disks – Part 4	213
Table 19: Technical data for hard disks – Part 5	214
Table 20: Technical data for hard disks – Part 6	215
Table 21: Technical data for hard disks – Part 7	216

Related publications

Please apply to your local office for ordering the manuals.

Reliant UNIX 5.45
Reliant UNIX Installation
Installation Guide

Target group
System administrators

Content
The manual describes the re-installation and updating of Reliant UNIX as well as initial installation of it when there is a new system disk.

Reliant UNIX 5.45
Reliant UNIX Operation
System Administrator's Guide

Target group
System administrators

Content
This manual describes the administration of Reliant UNIX.

RM600 E
(Reliant UNIX)
Operating Manual

Target group
Those responsible for setting up and operating the system

Content
Information about installing and starting up the system as well as maintenance and debugging.

RM600 E40/E80 CS42

(Reliant UNIX)

Operating Manual

Target group

Those responsible for setting up and operating the system

Content

Information about installing and starting up the system as well as maintenance and debugging.

Reliant UNIX 5.45

Commands. User's Reference Manual

Target group

Users, system administrators

Content

The manual describes the user commands of the Reliant UNIX 5.44 operating system.

Reliant UNIX 5.45

System Administrator's Reference Manual

Target group

System administrators

Content

The manual describes commands and application programs for system maintenance as well as file formats and special files for administration of the Reliant UNIX 5.45 operating system.

Reliant UNIX 5.44

System Administrator's Guide

User Guide

Target group

System administrators

Content

The manual provides an introduction to UNIX system administration. It describes, amongst other things, system administration tools, the configuration and support of UNIX systems, system accounting and aspects of system security. The manual is valid for both MIPS-based and Intel-based UNIX systems.

Reliant UNIX 5.45
Network Administration
System Administrator Guide

Target group
System administrators

Content
Network utilities and protocols such as TCP/IP, NFS and RFS.

Reliant UNIX 5.45
Networking Reference Manual

Target group
Programmers, system administrators, users

Content
The manual describes commands, functions, utilities and system files needed to work with computers which communicate in a network.

Reliant UNIX 5.45
Network Programming Interfaces
Programmer's Guide

Target group
Programmers

Content
This manual describes the TLI, sockets and RPC programming interfaces that Reliant UNIX makes available for the development of communication software.

Reliant UNIX 5.44
Virtual Disks
System Administrator's Guide

Target group
System administrators

Content
The manual describes the various types of virtual disks and how to configure them.

Reliant UNIX 5.45
Configuring Virtual Disks using VDisk Lite
System Administrator's Guide

Target group
System administrators

Content
This manual describes how to configure virtual disks using VDisk Lite. VDisk Lite provides a convenient user interface for configuring virtual disks. It manages disk areas and guards against errors in the configuration.

Reliant UNIX 5.45
VERITAS File System (VxFS) 2.1
System Administrator Guide

Target group
System administrators

Content
The manual describes the VxFS file system. It names diagnostic and error messages and explains what to do in each case.

Reliant UNIX 5.44
System Administration and Hardware Configuration Using the SYSADM Interface
System Administrator's Guide

Target group
System administrators

Content
The manual describes the administration and configuration of RM systems alphanumeric SYSADM user interface.

SINIX/windows User Environment V3.0
Introduction to Handling and Configuration
User Guide

Target group

Users with a basic knowledge of UNIX

Content

The manual introduces the components of SINIX/windows-Desktop. It illustrates how the interface elements can be used and how the desktop is configured interactively.

Other publications

P. E. Bregman S. A. Browning

UNIX System V Release 4 - Performance Management

Prentice Hall

ISBN 0-13-016429-1

B. Goodheart J. Cox

UNIX System V Release 4 - The Magic Garden Explained

Prentice Hall

ISBN 0-13-098138-9

B. Goodheart J. Cox

The internals of UNIX SYSTEM V Release 4

Prentice Hall

ISBN 0-13-091838-9

M. Loukides

System Performance Tuning

O'Reilly & Associates, Inc.

ISBN 0-937175-60-9

A. H. Majidmehr

Optimizing UNIX for Performance

Prentice Hall

ISBN 0-13-111551-0

R. C. Wilson

UNIX Benchmark and Test Suite Analysis

Prentice Hall

ISBN 0-13-125634-3

Index

- #BND (column) 82
- #DRVBND (column) 82
- #PINTBND (column) 82
- #XBND (column) 82, 134
- \$HOME/.profile (file) 143, 230
- \$HOME/measure/dk.dat (file) 70, 72, 117
- \$HOME/measure/dk-b.dat (file) 70, 74
- \$HOME/measure/dk-g.dat (file) 70, 74
- \$HOME/measure/dk-p.dat (file) 70, 74
- \$HOME/measure/ipcs_msg.dat (file) 70, 71, 127
- \$HOME/measure/ipcs_sem.dat (file) 70, 71, 128
- \$HOME/measure/ipcs_shm.dat (file) 70, 71, 130
- \$HOME/measure/ps.dat (file) 70, 71, 74, 86, 87, 105
- \$HOME/measure/sar_asc.dat (file) 70, 74
- \$HOME/measure/sar_bin.dat (file) 70, 73, 74
- \$HOME/measure/vm.dat (file) 70, 72, 92
- \$MANPATH (environment variable) 57
- \$PATH (environment variable) 57, 142, 143, 166, 232
- %busy (column) 115, 116, 206, 207, 208
- %idle (column) 36, 83, 84, 113, 114
- %iused (column) 123
- %rcache/s (column) 95, 119
- %runocc (column) 36, 85
- %sys (column) 36, 61, 83, 84, 113, 205, 206, 207
- %ufipf/s (column) 36
 - see also %uufsipf
- %uufsipf (column) 99
- %usr (column) 36, 61, 83, 84, 113, 205, 206, 207
- %wcache (column) 95, 119
- %wio (column) 36, 83, 84, 113, 114
- /dev (directory) 63
- /dev/pts (directory) 200
- /etc/.osm (file) 81, 90
- /etc/conf/bin (directory) 172
- /etc/conf/cf.d (directory) 163, 172
- /etc/conf/cf.d/mtune (file) 48, 49, 163, 164, 172, 181, 220
- /etc/conf/cf.d/stune (file) 48, 49, 163, 172, 180, 220
- /etc/conf/pack.d/kernel/space.c (file) 20, 22
- /etc/default/boot (file) 17, 18
- /etc/default/init (file) 167
- /etc/default/login (file) 167, 197
- /etc/default/simond (file) 58, 59
- /etc/dktab (file) 140
- /etc/vfstab (file) 25, 142
- /opt/monitor (directory) 57
- /opt/monitor/bin (directory) 57
- /stand/unix (file) 177
- /stand/unix.old (file) 179
- /usr/include/sys/vmparam.h (file) 15
- /usr/lib/sa/sa1 (file) 31
- /usr/lib/sa/sa2 (file) 31
- /usr/lib/X11/app-defaults/Simon (file) 58, 59
- /var/adm/log/messages (file) 81, 90
- /var/adm/sa/sadd (file) 31
- /var/adm/sa/sardd (file) 31
- /var/spool/cron/atjobs (directory) 150
- /var/spool/cron/crontabs (directory) 150
- 32-bit universe 11
- 64-bit universe 11

A

activity

- disks 69
- IPC 69
- memory 69
- page-in 33, 100
- page-out 33, 98
- paging 41, 69, 98, 102
- process 69, 71, 73
- queue 33
- swapping 33, 69, 102
- system 73

address **217**

- Ethernet 79
- field for virtual addresses 188

address space of kernel, virtual 10

AIO_HIWAT (kernel parameter) 198

AIO_LOWAT (kernel parameter) 198

AIO_MAX (kernel parameter) 199

AIO_VERSION (kernel parameter) 199

alignment 16, 20

allocation of swap area 108

allocation unit 25, 217

ARG_MAX (kernel parameter) 183

arp(1M) (command) 30

assigning a swap area 137

atch/s (column) 101

attributes of the CPU 82

ATTRS (column) 82

autoconf(8) (command) 29, 77, 80, 109

available main memory 90

avm (column) 40, 43, 93

avque (column) 115, 206, 207, 208

avserv (column) 116, 206, 207, 208

avwait (column) 116

B

BDFLUSHR (kernel parameter) 136, 165, 166, 183

BDM, see Board Debug Monitor

big page 16, **217**

for page tables 18

for shared memory data 17

for user text 18

blks/s (column) 116

block

indirect 183

inode 183

block number 185

blocks (column) 107

blocks available (column) 108

blocks used (column) 108

Board Debug Monitor 180, 217

boot mode 219

Boot1 218

Boot2 218

booting 180

booting **218**

Boot2 180

new kernel 178

old kernel 179, 180

SASH 179

bread/s (column) 36, 119

brk() (system call) 11

buffer **220**

for data 136, 183, 184, 185

for input/output 185

buffer cache 15, 95, 118, 136

new 10

old 10, 12

buffer head 184

buffer pool 220

BUFHWM (kernel parameter) 12, 13, 90, 95, 119, 165, 183

bwrit/s (column) 36, 119

C

cache

buffer 15, 95, 118, 136

DNLC 121, 147

page **15**

primary cache 6

second level cache 6

- cache coherent non-uniform memory
 - access 21
- cache memory, secondary 82
- CACHSZ (column) 82
- callout structure 184
- canch/s (column) 36
- CBYTES (column) 127, 128
- ccNUMA, see cache coherent non-uniform memory access
- Central Services & Interfaces 218
- channel, virtual 202
- class of a process 135, 195, 196
- CLOCK (column) 82, 209
- clock speed, see frequency of CPU
- clone interface 190
- cluster 139, 140
- COFF format 187
- column
 - #BND 82
 - #DRVBND 82
 - #PINTBND 82
 - #XBND 82, 134
 - %busy 115, 116, 206, 207, 208
 - %idle 36, 83, 84, 113, 114
 - %iused 123
 - %rcache/s 95, 119
 - %runocc 36, 85
 - %sys 36, 61, 83, 84, 113, 205, 206, 207
 - %ufipf/s 36
 - see also %ufsipf
 - %ufsipf 99
 - %usr 36, 61, 83, 84, 113, 205, 206, 207
 - %wcache 95, 119
 - %wio 36, 83, 84, 113, 114
 - atch/s 101
 - ATTRS 82
 - avm 40, 43, 93
 - avque 115, 206, 207, 208
 - avserv 116, 206, 207, 208
 - await 116
 - blks/s 116
 - blocks 107
 - blocks available 108
 - blocks used 108
 - bread/s 36, 119
 - bwrit/s 36, 119
 - CACHSZ 82
 - canch/s 36
 - CBYTES 127, 128
 - CLOCK 82, 209
 - CPU 81
 - cpu 40
 - cpu(s) 43
 - CPUTYPE 82
 - create/s 121
 - device 115
 - dirbk/s 120
 - dirblk/s 36
 - see also dirbk/s
 - exec/s 36, 124
 - file-sz 94, 125
 - filsz 36
 - see also file-sz
 - fork/s 36, 124
 - FP1 82
 - FP2 82
 - fr 102
 - fre 40, 43, 93
 - free 107
 - freem 36
 - see also freemem
 - freemem 15, 91, 205
 - frees 36
 - see also freeswp
 - freeswp 92
 - fst_mem 97
 - getpage/s 122
 - ifree 123
 - iget/s 36, 120
 - indsz 36
 - see also inod-sz
 - inod-sz 94, 125
 - int/s 83, 113, 206, 207, 209
 - intdef/s 83, 113
 - iused 123

lcksz 36
 see also lock-sz
lg_mem 97
lock-sz 94, 125
lookup/s 121
LPID 130
lread/s 36, 119
lwrit/s 36, 119
mdmin/s 36
memory 40, 43
msg/s 36, 126, 206, 207
namei/s 36, 120
NATTCH 130
NSEMS 129
open/s 121
other/s 122
OTIME 129
outch/s 36
ov 36, 94, 125
ovsz_mem 97
page 40, 43
PCPU 81
pflt/s 36, 101
pgfree/s 36, 99
pgin/s 36, 101, 102
pgout/s 36, 99
pgscan/s 36, 99
pi 102
po 102
ppgin/s 101, 205, 206, 208, 209
ppgout/s 36, 99, 100, 205, 206,
 207, 209
ppin/s 36
 see also ppgin/s
prcsz 36
 see also proc-sz
pread/s 36, 61, 119
procs 39, 42
proc-sz 94, 125
pswch/s 36, 84, 103, 206, 207
pswin/s 36, 103
pswot/s 36, 103
putpage/s 122
pwrit/s 36, 61, 119

QBYTES 127
QNUM 127, 128
r+w/s 116, 206, 207, 208, 209
rchar/s 36, 124
rcvin/s 36
readdir/s 122
RSS 88, 106, 205
runq-sz 36, 85, 205, 206, 207
s0xx 118
scall/s 36, 124
SEGSZ 130, 131
sema/s 36, 126, 206, 207
shmем 36
si 104
SIZE 88, 104, 106
slck/s 36
 see also slock/s
slock/s 101
sml_mem 97
so 104
sr 102
sread/s 36, 124
STATUS 81
swpin/s 36, 103
swpot/s 36, 103
swrit/s 36, 124
SZ 107
TIME 88, 106, 205
vflt/s 36, 101, 102, 207, 208
wchar/s 36, 124
wspick 83
xmtin/s 36
command **220**
 arp(1M) 30
 autoconf(8) 29, 77, 80, 109
 crash(1M) 29, 48, 186
 df(1) 29, 61, 74, 112, 122
 dfspace 29, 112
 dispadmín(1M) 29, 134, 135
 dkconfig(8) 29
 dkmap(8) 29, 111
 dkpart(8) 29, 110
 dkstat(1M) 29, 37, 69, **72**, 117
 dkstat(1M) -b 37, 38

dkstat(1M) -g 37, 38
dkstat(1M) -p 37, 38
dkstat(1M) -z 37
du(1) 143
etherstat(1M) 30
find(1) 144, 145
idbuild(1M) **177**, 180, 225
id tune(1M) 149, 172, **174**, 180
ifconfig(1M) 30
ikdb(8) 48, 186
iotail(1M) **47**
iotail(1M) -m 47
iotail(1M) -n 47
iotail(1M) -p 47
iotail(1M) -t 47
iotail(1M) -v 47
iotail(1M) -y 47
ipcs(1) 29, 69
launchit(1M) 134, 167
mkfs(1M) 147
mpcntl(1M) 29, 81, **133**, 133, 134
mpstat(1M) 29, 39, 69, 104
netstat(1M) 30
nfsstat(1) 30
nice(1) 134
pagesize(1) 8, 29, 91
pfenable(1M) 167
ping(1M) 30
preload(8) 18
prf(7) 29
priocntl(1) 29, 134
proceso(1M) 29, 51, 104
proceso(1M) -c 52
proceso(1M) -k 52
proceso(1M) -M 52
proceso(1M) -p 52
proceso(1M) -r 52
proceso(1M) -s 52
proceso(1M) -u 52
proceso(1M) -uu 52
ps(1) 29, 69, 71, 87, 151
quickdd(1M) 23
ruptime(1) 30
sadc(1M) 29
sar(1) 30, **31**, 69
sar(1) -A 32, 74
sar(1) -a 32, 35, 120, 146
sar(1) -b 32, 35, 61, 95, 118
sar(1) -c 32, 33, 35, 45, 123
sar(1) -d 32, 33, 35, 45, 114, 207, 208
sar(1) -g 32, 33, 35, 98, 206, 207
sar(1) -K 32, 96
sar(1) -k 35
sar(1) -l 32
sar(1) -m 32, 35, 45, 126, 128, 206, 207
sar(1) -P 32, 45, 83
sar(1) -p 32, 33, 35, 45, 100, 206, 208
sar(1) -q 32, 33, 35, 45, 61, 84, 206, 207
sar(1) -r 32, 33, 35, 61
sar(1) -u 32, 33, 35, 61, 113, 206, 207
sar(1) -v 32, 35, 93, 125
sar(1) -w 32, 33, 35, 103, 206, 207
sar(1) -x 32, 121
sar(1) -y 32, 35
scstat(8) 30, **45**, 69, 124
scstat(8) -a 45
scstat(8) -c 46
scstat(8) -d 45
scstat(8) -p 46
scstat(8) -z 45
simon(1) 30, 33, 57, 58, 59
simond(1) 57, 58
spray(1M) 30
strings(1) 171
swap(1M) 30, 108, 112
sysadm(1M) 34, 233
sysdef(1M) 30, 168
tcpdump(1) 30
timex(1) 30
truss(1) 30, 89
tunefs(1M) 148
tunestat(1M) **48**

- ulimit(1) 167, 196, 197
- uname(1) 66, 80
- uptime(1) 30, 44, 55
- vmstat(1M) 30, 41, 69, **72**, 92, 102
- vmstat(1M) -d 42
- vmstat(1M) -f 42
- vmstat(1M) -m 42
- vmstat(1M) -S 42
- vmstat(1M) -s 42
- who(1) 178
- xload 33, 55
- computer
 - configuration **220**
 - name 80
 - type 80
- concatenated disk, virtual 25, 139
- configuration directory 163, 172
- configuration of computers **220**
- configuring the kernel 163–203, 221
- console **221**
- control field 192
- controller **221**
 - Ethernet 79
- controllers, number of 12
- copy on write 101
- core (file) 196, 221
- core image 186, **221**
- CPU 6
 - attributes 82
 - frequency 82, 206, 207, 209
 - load 33
 - logical number 81
 - number 80
 - physical number 81
 - queue 55, 85
 - status flags 81
 - type 82
 - waiting for input/output 113
- CPU (column) 81
- cpu (column) 40
- CPU load 69
- cpu(s) (column) 43
- CPUTYPE (column) 82
- crash **221**
- crash(1M) (command) 29, 48, 186
- create/s (column) 121
- CSI, see Central Services & Interfaces
- cylinder **222**
- D**
- daemon **222**
 - pageout 15, 41, 43, 91, 98, 102
 - sched 15
- daemon processes, unneeded 151
- data buffer 136, 183, 184, 185
- deadlocks 189, 201
- delayed interrupt 83, 113
- demand paging **222**
- DEN_HWM_PAGES (kernel parameter) 202
- DEN_LWM_PAGES (kernel parameter) 202
- detail windows of SINIX/monitor 62
- device **222**
- device (column) 115
- device driver parameter 194
- device nodes 183
- device number 183, 185
- df(1) (command) 29, 61, 74, 112, 122
- dfspace (command) 29, 112
- dirbk/s (column) 120
- dirblk/s (column) 36
 - see also dirbk/s
- directory **222**
 - /dev 63
 - /dev/pts 200
 - /etc/conf/bin 172
 - /etc/conf/cf.d 163, 172
 - /opt/monitor 57
 - /opt/monitor/bin 57
 - /var/spool/cron/atjobs 150
 - /var/spool/cron/crontabs 150
- directory entry **222**
- directory name lookup cache 26, 121, 147, 185
- directory tree **222**
- disk activity 33, 69

- disk, virtual 24, 110, 138, 201, 235
- dispadm(1M) (command) 29, 134, 135
- dispatcher table 41, 135
- dk_disk_eval* (procedure) 70, 117
- dk_meas* (procedure) 70, 72
- dkconfig(8) (command) 29
- dkmap(8) (command) 29, 111
- dkpart(8) (command) 29, 110
- dkstat statistics 74, 117
- dkstat(1M) (command) 29, 37, 69, 72, 117
 - b 37, 38
 - g 37, 38
 - p 37, 38
 - z 37
- DNLC, see directory name lookup cache
- double mismatch 19
- DPTGCHANS (kernel parameter) 202
- driver, log 190
- du(1) (command) 143
- dump, see core image
- DVHANGUP (kernel parameter) 201
- dynamic libraries 187
- dynamic system table 149
- E**
- EFBIG (error) 197
- EHIOS, see Enhanced High Performance Input Output System
- ELIBMAX (error) 187
- EMFILE (error) 197
- Enhanced High Performance Input Output System 218
- ENODEV (error) 200
- ENOMEM (error) 197, 198
- ENOSR (error) 189, 191
- environment **223**
- environment variable
 - \$MANPATH 57
 - \$PATH 57, 142, 143, 166, 232
- ENXIO (error) 190
- error **223**
 - EFBIG 197
 - ELIBMAX 187
 - EMFILE 197
 - ENODEV 200
 - ENOMEM 197, 198
 - ENOSR 189, 191
 - ENXIO 190
- Ethernet address 79
- Ethernet controller 79
- etherstat(1M) (command) 30
- exec() (system call) 223
- exec/s (column) 36, 124
- external frequency of CPU 209
- F**
- field for virtual addresses 188
- file
 - \$HOME/.profile* 143, 230
 - \$HOME/measure/dk.dat* 70, 72, 117
 - \$HOME/measure/dk-b.dat* 70, 74
 - \$HOME/measure/dk-g.dat* 70, 74
 - \$HOME/measure/dk-p.dat* 70, 74
 - \$HOME/measure/ipcs_msg.dat* 70, 71, 127
 - \$HOME/measure/ipcs_sem.dat* 70, 71, 128
 - \$HOME/measure/ipcs_shm.dat* 70, 71, 130
 - \$HOME/measure/ps.dat* 70, 71, 74, 86, 87, 105
 - \$HOME/measure/sar_asc.dat* 70, 74
 - \$HOME/measure/sar_bin.dat* 70, 73, 74
 - \$HOME/measure/vm.dat* 70, 72, 92
 - /etc/osm* 81, 90
 - /etc/conf/cf.d/mtune* 48, 49, 163, 164, 172, 181, 220
 - /etc/conf/cf.d/stune* 48, 49, 163, 172, 180, 220
 - /etc/conf/pack.d/kernel/space.c* 20, 22
 - /etc/default/boot* 17, 18

- /etc/default/init* 167
- /etc/default/login* 167, 197
- /etc/default/simond* 58, 59
- /etc/dktab* 140
- /etc/vfstab* 25, 142
- /stand/unix* 177
- /stand/unix.old* 179
- /usr/include/sys/vmparam.h* 15
- /usr/lib/sa/sa1* 31
- /usr/lib/sa/sa2* 31
- /usr/lib/X11/app-defaults/Simon* 58, 59
- /var/adm/log/messages* 81, 90
- /var/adm/sa/sadd* 31
- /var/adm/sa/sardd* 31
- core* 196, 221
- owner* 187
- file system **24**
 - VxFS 24
- file system parameter 188
- file-sz (column) 94, 125
- filsz (column) 36
 - see also file-sz
- find(1) (command) 144, 145
- firmware monitor 179
- FLCKREC (kernel parameter) 173, 183
- floating point interrupt 41
- fork() (system call) 224
- fork/s (column) 36, 124
- FP1 (column) 82
- FP2 (column) 82
- FPU owner rights 41
- FPU type 82
- fr (column) 102
- fre (column) 40, 43, 93
- free (column) 107
- free list 200
- freem (column) 36
 - see also freemem
- freemem (column) 15, 91, 205
- frees (column) 36
 - see also freeswp
- freeswp (column) 92
- frequency of CPU 82, 206, 207, 209
- fsflush (process) 136
- fst_mem (column) 97
- G**
 - generating a kernel 177
 - getpage/s (column) 122
 - glossary 217–235
 - GPGSLO (kernel parameter) 15, 102, 189
 - group **224**
 - group ID 187, **224**
 - group permissions 185
- H**
 - hard disk 24, **225**
 - installing additional hard disk 148
 - partition layout 110
 - hash entry 13
 - hash list 186
 - hash table 185, 190
 - HCORLIM (kernel parameter) 196
 - HCPULIM (kernel parameter) 197
 - HDATLIM (kernel parameter) 197
 - heap 197
 - HFNOLIM (kernel parameter) 166, 197
 - HFSZLIM (kernel parameter) 166, 198
 - High Performance Input Output System 218
 - high resolution timer 195
 - HIOS, see High Performance Input Output System
 - histogram 55
 - hit rate 95
 - HRTIME (kernel parameter) 173, 195
 - hrtimes structure 195
 - HRVTIME (kernel parameter) 173, 195
 - HSTKLIM (kernel parameter) 198
 - HVMMLIM (kernel parameter) 198

I

Id Structure 192
 idbuild(1M) (command) **177**, 180, 225
 IDLE_MIGRATE (kernel parameter) 201
 idtune(1M) (command) 149, 172, **174**, 180
 ifconfig(1M) (command) 30
 ifree (column) 123
 iget/s (column) 36, 120
 ikdb(8) (command) 48, 186
 i-list 26, 94, 125, 149, **225**
 indirect block 183
 indsz (column) 36
 see also inod-sz
 infinite loop 89
 inode 15, **225**
 number of inodes in VxFS file system 24, 25
 inode block 183
 inode number **225**
 inod-sz (column) 94, 125
 input/output buffer 185
 input/output statistics 73
 input/output, CPU waiting 113
 int/s (column) 83, 113, 206, 207, 209
 intdef/s (column) 83, 113
 intent log 25, 225
 interface
 clone 190
 internal frequency of CPU 206, 207
 interprocess communication 27, 166, 191
 interrupt 83, 113, 206, 207, 209
 delayed 83, 113
 floating point 41
 iotail(1M) (command) **47**
 -m 47
 -n 47
 -p 47
 -t 47
 -v 47
 -y 47

iotail(1M) (Kommando) 29
 IPC activity 69
 IPC parameter 191
 ipcs(1) (command) 29, 69
 itimes structure 195
 iused (column) 123

K

kernel **225**
 booting 178, 179, 180
 configuring 163–203, 221
 generating 177
 messages 167
 multiprocessor 177
 occupied main memory 90
 single-processor 177
 size 180
 virtual address space 10
 kernel memory allocation 96
 kernel messages 167
 kernel parameter **26**
 default value 163
 description 181
 maximum value 163
 minimum value 163
 modifying 172
 see also types of kernel parameter
 suitable for tuning measures 165
 tunable 163
 AIO_HIWAT 198
 AIO_LOWAT 198
 AIO_MAX 199
 AIO_VERSION 199
 ARG_MAX 183
 BDFLUSHR 136, 165, 166, 183
 BUFHWM 12, 13, 90, 95, 119, 165, 183
 DEN_HWM_PAGES 202
 DEN_LWM_PAGES 202
 DPTGCHANS 202
 DVHANGUP 201
 FLCKREC 173, 183
 GPGSLO 15, 102, 189
 HCORLIM 196

HCPULIM 197
HDATLIM 197
HFNOLIM 166, 197
HFSZLIM 166, 198
HRTIME 173, 195
HRVTIME 173, 195
HSTKLIM 198
HVMMMLIM 198
IDLE_MIGRATE 201
KM_CACHE_* 199
KM_OSB_HWM 200
KM_OSB_SIZE 200
KV_SIZE 11, 12, 90, 96, 173, 200
MAXCLSYPRI 195
MAXMINOR 183
MAXNETUSERS 173, 202
MAXNFSCLIENTS 202
MAXPMEM 183
MAXUP 165, 166, 184
MAXUSERS 13, 165, 166, 173, 203
MAXVDUNIT 201
MINARMEM 189
MINASMEM 167, 189
MSGMAP 191
MSGMAX 191
MSGMNB 128, 191
MSGMNI 26, 127, 192
MSGSEG 128, 192
MSGSSZ 128, 192
MSGTQL 128, 192
NAUTOPUSH 167, 189
NAUTOUP 136, 165, 166, 184
NBUF 13, 184
NC_NAMLEN 184
NCALL 168, 173, 184
NCPYRIGHT 194
NCSIZE 26, 90, 121, 147, 185
NDQUOT 167, 173, 188
NGROUPS_MAX 185
NHBUF 13, 185
NINODE 90, 165, 173, 188
NLOG 167, 190
NPBUF 185
NPROC 12, 13, 26, 90, 165, 166, 167, 173, 186
NPROCHASH 168, 186
NPTEM 165, 167, 200
NPTY 165, 167, 200
NRNODE 173, 188
NSTRPHASH 190
NSTRPUSH 190
NTCP 173
NUDP 173
NUMSAD 190
NUMSOM 173
NUMSXT 194
NUMTIM 173
NUMTRW 173
PAGEIOPOOL_HWM 186
PAGEIOPOOL_LWM 186
PAGEOUT_DELTA 15, 189
PPNUMLOCK 201
PRFMAX 194
PUTBUFSZ 186
QUEUED_LOCKS 203
RBUFPOOL_HWM 187
RBUFPOOL_LWM 187
RSTCHOWN 187
SCORLIM 196
SCPULIM 197
SDATLIM 107, 167, 197
SDILARGEIO 187
SECURE_FS 203
SEMAEM 192
SEMAP 192
SEMMNI 129, 192
SEMMNS 26, 129, 193
SEMMNU 193
SEMMSL 129, 193
SEMOPM 193
SEMUME 193
SEMVMX 193
SFNOLIM 165, 166, 168, 197
SFSZLIM 166, 167, 197
SHLBMAX 167, 187
SHMMAX 131, 193
SHMMIN 194

-
- SHMMNI 26, 131, 194
 - SHMSEG 131, 194
 - SM_SIZE 14, 90, 201
 - SPTMAP 188
 - SSTKLIM 107, 198
 - STRCTLSZ 190
 - STRMSGSZ 190
 - STRTHRESH 167, 191
 - SVMMLIM 107, 167, 198
 - SYSTEM_UMASK 201
 - TAPE_NO_AUTOLOAD 203
 - TCPDEBUG 173
 - TPCLOCK 195
 - TSMAXUPRI 195
 - TSSLEEPDECAY 196
 - TSSLEEPMAX 196
 - TSSLEEPWAIT 196
 - TSSLPSWPDIV 196
 - UFSINODE 99
 - UFSNINODE 12, 26, 90, 149, 168, 173, 188
 - kernel spin locks 83
 - KM_CACHE_* (kernel parameter) 199
 - KM_OSB_HWM (kernel parameter) 200
 - KM_OSB_SIZE (kernel parameter) 200
 - KMA, see kernel memory allocation 96
 - Kommando
 - iotail(1M) 29
 - KPTESEG (segment) 19
 - KV_SIZE (kernel parameter) 11, 12, 90, 96, 173, 200

 - L**
 - LAN Manager 84
 - launchit(1M) (command) 134, 167
 - layout (hard disk) 110
 - lcksz (column) 36
 - see also lock-sz
 - lg_mem (column) 97

 - libraries
 - dynamic 187
 - static 187
 - link **226**
 - list, hash 186
 - load
 - CPU 33, 69
 - disks 33
 - main memory 33
 - lock table 94, 125
 - lock-sz (column) 94, 125
 - log driver 190
 - logical number of CPU 81
 - login 203, **226**
 - login name root 231, 233
 - lookup/s (column) 121
 - LOTSFREE (parameter) 15, 91, 98
 - LPID (column) 130
 - lread/s (column) 36, 119
 - lwrit/s (column) 36, 119

 - M**
 - main memory 7, 80
 - alignment of shared memory 16, 20
 - available 90
 - characteristic values 90
 - extending 136
 - load 33
 - occupied by kernel 90
 - physical 90, 183, 226
 - size of 12, 80
 - virtual 92
 - main window of SINIX/monitor 60
 - main_meas* (procedure) 70, 74, 75
 - manual, notational conventions 3
 - Master Test Handler 226
 - MAXCLSYSPRI (kernel parameter) 195
 - MAXMINOR (kernel parameter) 183
 - MAXNETUSERS (kernel parameter) 173, 202
 - MAXNFSCLIENTS (kernel parameter) 202

- MAXPMEM (kernel parameter) 183
 - MAXUP (kernel parameter) 165, 166, 184
 - MAXUSERS (kernel parameter) 13, 165, 166, 173, 203
 - MAXVDUNIT (kernel parameter) 201
 - mdmin/s (column) 36
 - memory **226**
 - see also main memory
 - see also shared memory
 - memory (column) 40, 43
 - memory activity 69
 - memory disk 79, 80, **227**
 - memory freelist 93, 101
 - memory image, see core image
 - memory management, virtual 8, 235
 - memory mapped file 14
 - memory statistics 73
 - message header 192
 - message parameter 191
 - message queues 26, 27, 71, 127, 166, 192, **227**
 - message segment 191, 192
 - MINARMEM (kernel parameter) 189
 - MINASMEM (kernel parameter) 167, 189
 - mirror disk, virtual 139, 227
 - mismatch 19
 - see also double mismatch 19
 - mkfs(1M) (command) 147
 - mmap() (system call) 14
 - mode
 - system 83, 113, 206, 207
 - timesharing 196
 - user 83, 113, 206, 207
 - mpcntl(1M) (command) 29, 81, **133**, 133, 134
 - mpstat(1M) (command) 29, 39, 69, 104
 - msg/s (column) 36, 126, 206, 207
 - MSGMAP (kernel parameter) 191
 - MSGMAX (kernel parameter) 191
 - MSGMNB (kernel parameter) 128, 191
 - MSGMNI (kernel parameter) 26, 127, 192
 - MSGSEG (kernel parameter) 128, 192
 - MSGSSZ (kernel parameter) 128, 192
 - MSGTQL (kernel parameter) 128, 192
 - MTH, see Master Test Handler
 - multiprocessor kernel 177
- N**
- name
 - of computer 80
 - of operating system 80
 - of partitions 141
 - of virtual disk 141
 - of virtual disk area 141
 - of virtual mirror disk 141
 - namei/s (column) 36, 120
 - NATCH (column) 130
 - NAUTOPUSH (kernel parameter) 167, 189
 - NAUTOUP (kernel parameter) 136, 165, 166, 184
 - NBUF (kernel parameter) 13, 184
 - NC_NAMLEN (kernel parameter) 184
 - NCALL (kernel parameter) 168, 173, 184
 - NCPYRIGHT (kernel parameter) 194
 - NCSIZE (kernel parameter) 26, 90, 121, 147, 185
 - NDQUOT (kernel parameter) 167, 173, 188
 - netstat(1M) (command) 30
 - new buffer cache 10
 - nfsstat(1) (command) 30
 - NGROUPS_MAX (kernel parameter) 185
 - NHBUF (kernel parameter) 13, 185
 - nice(1) (command) 134
 - NINODE (kernel parameter) 90, 165, 173, 188

-
- NLOG (kernel parameter) 167, 190
 - node name of computer 80
 - notational conventions of manual 3
 - NPBUF (kernel parameter) 185
 - NPROC (kernel parameter) 12, 13, 26, 90, 165, 166, 167, 173, 186
 - NPROCHASH (kernel parameter) 168, 186
 - NPTM (kernel parameter) 165, 167, 200
 - NPTY (kernel parameter) 165, 167, 200
 - NRNODE (kernel parameter) 173, 188
 - NSEMS (column) 129
 - NSTRPHASH (kernel parameter) 190
 - NSTRPUSH (kernel parameter) 190
 - NTCP (kernel parameter) 173
 - NUDP (kernel parameter) 173
 - null device 227
 - number
 - block 185
 - device 183, 185
 - of controllers 12
 - of CPUs 80
 - of processors 12, 80
 - processes for users 184
 - number of inodes in VxFS file system 24, 25
 - NUMSAD (kernel parameter) 190
 - NUMSOM (kernel parameter) 173
 - NUMSXT (kernel parameter) 194
 - NUMTIM (kernel parameter) 173
 - NUMTRW (kernel parameter) 173
- O**
- old buffer cache 10, 12
 - open/s (column) 121
 - operating system **228**
 - name 80
 - performance 1, 229
 - revision number 80
 - version 80
 - operating system profiler 29, 194
 - other/s (column) 122
 - OTIME (column) 129
 - outch/s (column) 36
 - ov (column) 36, 94, 125
 - overhead 16
 - ovsz_mem (column) 97
 - owner of file 187
- P**
- package
 - Sldptg2 202
 - Slmon 57
 - page
 - see also bigp page
 - page (column) 40, 43
 - page cache **15**
 - page fault 100, 101, 222, **228**
 - page of virtual main memory **8, 228**
 - page table 19
 - page, see page of virtual main memory
 - page-in activity 33, 100
 - PAGEIOPOOL_HWM (kernel parameter) 186
 - PAGEIOPOOL_LWM (kernel parameter) 186
 - pageout (daemon) 15, 41, 43, 91, 98, 102
 - page-out activity 33, 98
 - PAGEOUT_DELTA (kernel parameter) 15, 189
 - pagesize(1) (command) 8, 29, 91
 - paging **9, 24, 136, 228**
 - paging activity 41, 69, 98, 102
 - paging parameter 189
 - parameter LOTSFREE 15, 91, 98
 - partition layout 110
 - partition table **229**
 - partitions **229**
 - size 110
 - pathname **229**
 - PCPU (column) 81

- performance of operating system
 - 1, 229
- permissions **229**
- pfenable(1M) (command) 167
- pflt/s (column) 36, 101
- pgfree/s (column) 36, 99
- pgin/s (column) 36, 101, 102
- pgout/s (column) 36, 99
- pgscan/s (column) 36, 99
- physical main memory 90, 183, 226
- physical number of CPU 81
- pi (column) 102
- PID 62, 63, 130, 186
- ping(1M) (command) 30
- pipe **229**
- po (column) 102
- ppgin/s (column) 101, 205, 206, 208, 209
- ppgout/s (column) 36, 99, 100, 205, 206, 207, 209
- ppin/s (column) 36
 - see also ppgin/s
- PPNUMLOCK (kernel parameter) 201
- prcsz (column) 36
 - see also proc-sz
- pread/s (column) 36, 61, 119
- preload(8) (command) 18
- prf(7) (command) 29
- PRFMAX (kernel parameter) 194
- primary cache 6
- priocntl(1) (command) 29, 134
- proc table 186
- procedure
 - dk_disk_eval* 70, 117
 - dk_meas* 70, 72
 - main_meas* 70, 74, 75
 - ps_meas* 70, 71
 - ps_size_eval* 70, 105
 - ps_stat_eval* 70, 86
 - ps_time_eval* 70, 87
 - vm_meas* 70, 72
 - vm_mem_eval* 70, 92
- proceso(1M) (command) 29, 51, 104
 - c 52
 - k 52
 - M 52
 - p 52
 - r 52
 - s 52
 - u 52
 - uu 52
- process 7, **230**
 - fsflush 136
 - sleeping 196
 - system status 85
 - unnneeded daemon process 151
 - uucp 150
- process activity 69, 71, 73
- process class 135, 195, 196
- process ID 62, 63, 130, 186, **230**
- process queue statistics 84, 86
- process scheduling 89, 134, 196
- process segment 11
- process switching 33, 41, 84
- process table 26, 94, 125, 186
- processes, number for users 184
- processors **7**
 - number of 12, 80
 - type 80
- procs (column) 39, 42
- proc-sz (column) 94, 125
- profile **230**
- program **230**
- Prom Monitor 179, **230**
- protection faults 101
- ps(1) (command) 29, 69, 71, 87, 151
- ps_meas* (procedure) 70, 71
- ps_size_eval* (procedure) 70, 105
- ps_stat_eval* (procedure) 70, 86
- ps_time_eval* (procedure) 70, 87
- pseudo-TTY 200
- pswch/s (column) 36, 84, 103, 206, 207
- pswin/s (column) 36, 103
- pswot/s (column) 36, 103
- ptem structure 200

-
- PUTBUFSZ (kernel parameter) 186
 - putpage/s (column) 122
 - pwrit/s (column) 36, 61, 119
 - Q**
 - QBYTES (column) 127
 - QNUM (column) 127, 128
 - queue **230**
 - for messages 192
 - of CPU 55, 85
 - queue activity 33
 - queue statistics for processes 84, 86
 - queued locks 203
 - QUEUED_LOCKS(kernel parameter) 203
 - quickdd(1M) (command) 23
 - quiet mode 48
 - quota table for the UFS file system 188
 - R**
 - r+w/s (column) 116, 206, 207, 208, 209
 - RAID, see Redundant Array of Independent Disks
 - RAID0 25, 139
 - RAID1 139
 - raw mode 114, 119, 187, 231
 - RBUFPOOL_HWM (kernel parameter) 187
 - RBUFPOOL_LWM(kernel parameter) 187
 - rchar/s (column) 36, 124
 - rcvin/s (column) 36
 - read() (system call) 14
 - readdir/s (column) 122
 - real-time clock 195
 - Redundant Array of Independent Disks 138
 - Remote Procedure Call 202
 - resource limits, parameters 196
 - response time 231
 - revision number of operating system 80
 - rlimit structure 181, 196
 - parameter 196
 - rnode structure 188
 - root
 - authorization 233
 - login name **231**, 233
 - permissions 231
 - root directory **231**
 - round robin algorithm 17, 22
 - RPC, see Remote Procedure Call
 - RSS (column) 88, 106, 205
 - RSTCHOWN (kernel parameter) 187
 - run queue 85
 - runq-sz (column) 36, 85, 205, 206, 207
 - ruptime(1) (command) 30
 - S**
 - s0xx (column) 118
 - SAD, see STREAMS Administration Driver
 - sadc(1M) (command) 29
 - sar measurement 114
 - sar(1) (command) 30, **31**, 69
 - A 32, 74
 - a 32, 35, 120, 146
 - b 32, 35, 61, 95, 118
 - c 32, 33, 35, 45, 123
 - d 32, 33, 35, 45, 114, 207, 208
 - g 32, 33, 35, 98, 206, 207
 - K 32, 96
 - k 35
 - l 32
 - m 32, 35, 45, 126, 128, 206, 207
 - P 32, 45, 83
 - p 32, 33, 35, 45, 100, 206, 208
 - q 32, 33, 35, 45, 61, 84, 206, 207
 - r 32, 33, 35, 61
 - u 32, 33, 35, 61, 113, 206, 207
 - v 32, 35, 93, 125
 - w 32, 33, 35, 103, 206, 207
 - x 32, 121
 - y 32, 35

- SASH **231**
 - booting 179
- sbrk() (system call) 11
- scall/s (column) 36, 124
- sched (daemon) 15
- scheduler 134, 196, 231
- scheduler parameter 195
- scheduling of processes 134, 196
- SCORLIM (kernel parameter) 196
- SCPULIM (kernel parameter) 197
- scstat(8) (command) 30, **45**, 69, 124
 - a 45
 - c 46
 - d 45
 - p 46
 - z 45
- SDATLIM (kernel parameter) 107, 167, 197
- SDILARGEIO (kernel parameter) 187
- search path of shell 232
- second level cache 6
- secondary cache memory 82
- SECTION 164
- sections of file */etc/conf/cf.d/mtune* 164
- secure file system 203
- SECURE_FS (kernel parameter) 203
- seg_kmem (segment) 10, 11
- segmap (segment) 10, 14, 118, 201
- segment
 - KPTESEG 19
 - process segment 11
 - seg_kmem 10, 11
 - segmap 10, 14, 118, 201
 - segu 10, 13
 - shared memory segment 16, 17, 193
 - user segment 10, 18
- SEGSZ (column) 130, 131
- segu (segment) 10, 13
- sem_undo structure 193
- sema/s (column) 36, 126, 206, 207
- SEMAEM (kernel parameter) 192
- semaphore 26, 27, 71, 114, 129, 166, 186, 193, **232**
- semaphore group 129, 192, 193
- semaphore operation 193
- semaphore parameter 192
- SEMMAP (kernel parameter) 192
- SEMMNI (kernel parameter) 129, 192
- SEMMNS (kernel parameter) 26, 129, 193
- SEMMNU (kernel parameter) 193
- SEMMSL (kernel parameter) 129, 193
- SEMOPM (kernel parameter) 193
- SEMUME (kernel parameter) 193
- SEMVMX (kernel parameter) 193
- set userid bit 181, **232**
- SFNOLIM (kernel parameter) 165, 166, 168, 197
- SFSZLIM (kernel parameter) 166, 167, 197
- shared memory 26, 27, 71, 130, 166, **232**, 232
 - alignment 16, 20
 - reducing 137
- shared memory parameter 193
- shared memory segment 16, 17, 131, 193
- shell **232**
 - search path 232
- shell script **232**
- SHLBMAX (kernel parameter) 167, 187
- shmat(2) (system call) 17
- shmем (column) 36
- shmget(2) (system call) 17
- SHMMAX (kernel parameter) 131, 193
- SHMMIN (kernel parameter) 194
- SHMMNI (kernel parameter) 26, 131, 194
- SHMSEG (kernel parameter) 131, 194
- si (column) 104

-
- Sldptg2 (package) 202
 - signal **233**
 - SIGPFE 41
 - SIGSEGV 198
 - SIGXCPU 197
 - SIGXFSX 197
 - SIGPFE signal 41
 - SIGSEGV signal 198
 - SIGXCPU signal 197
 - SIGXFSX signal 197
 - Slmon (package) 57
 - simon(1) (command) 30, 33, 57, 58, 59
 - simond(1) (command) 57, 58
 - simple disk, virtual 138
 - single-processor kernel 177
 - SINIX, revision number 80
 - SINIX/monitor 30, 57–67
 - detail windows 62
 - main window 60
 - SINIX/windows User Environment 55, 57
 - SINIX-specific parameters 198
 - size
 - of kernel 180
 - of main memory 12, 80
 - of partitions 110
 - SIZE (column) 88, 104, 106
 - SLC, see second level cache
 - slock/s (column) 36
 - see also slock/s
 - sleeping process 196
 - slock/s (column) 101
 - SM_SIZE (kernel parameter) 14, 90, 201
 - small code model 10
 - sml_mem (column) 97
 - so (column) 104
 - SPbus, see Synchronous Pipelined Bus
 - special file **233**
 - spinlocks 203
 - spray(1M) (command) 30
 - SPTMAP (kernel parameter) 188
 - sr (column) 102
 - sread/s (column) 36, 124
 - SSTKLIM (kernel parameter) 107, 198
 - stack area 198
 - statesave disk, virtual 139
 - static libraries 187
 - static system table 149
 - statistics
 - dkstat 74, 117
 - input/output 73
 - memory 73
 - process queues 84, 86
 - system call 45, 69
 - STATUS (column) 81
 - status flags of the CPU 81
 - status partition of mirror disk 141
 - STRCTL SZ (kernel parameter) 190
 - STREAMS Administrative Driver 190
 - STREAMS parameter 189
 - strings(1) (command) 171
 - striped disk, virtual 25, 38, 139, 149
 - STRMSG SZ (kernel parameter) 190
 - STRTHRESH (kernel parameter) 167, 191
 - structure
 - callout 184
 - hrtimes 195
 - itimes 195
 - pitem 200
 - rlimit 181, 196
 - rnode 188
 - sem_undo 193
 - superuser 167
 - see system administrator
 - SVMMLIM (kernel parameter) 107, 167, 198
 - swap area 92, 98, 107, 196, **233**
 - allocation 108
 - assignment 137
 - swap(1M) (command) 30, 108, 112
 - swapping **9**, 24, 136, **233**
 - swapping activity 33, 69, 102

- switching
 - to the Board Debug Monitor 180
 - to the firmware monitor 179
 - to the Prom Monitor 179
- swpin/s (column) 36, 103
- swpot/s (column) 36, 103
- swrit/s (column) 36, 124
- sync rate 166, 184
- Synchronous Pipelined Bus 78
- sysadm(1M) (command) 34, 233
- SYSADM, user interface 34–36, 57, **233**
- sysdef(1M) (command) 30, 168
- sysmips() (system call) 16
- system activity 73
- system administrator **233**
- system call 33, 84, **234**
 - brk() 11
 - exec() 223
 - fork() 224
 - mmap() 14
 - read() 14
 - sbrk() 11
 - shmat(2) 17
 - shmget(2) 17
 - sysmips() 16
 - write() 14
- system call statistics 45, 69
- system clock 184
- system disk **234**
- system load 55
- system mode 83, 113, 206, 207
- system status of processes 85
- system table
 - dynamic 149
 - static 149
- SYSTEM_UMASK (kernel parameter) 201
- SZ (column) 107
- T**
- table
 - dispatcher 135
 - dynamic system 149
 - hash 185, 190
 - i-list 26, 94, 125, 149
 - lock 94, 125
 - proc 186
 - process 26, 94, 125, 186
 - static system 149
- TAPE_NO_AUTOLOAD (kernel parameter) 203
- TCPDEBUG (kernel parameter) 173
- tcpdump(1) (command) 30
- text file **234**
- throughput **234**
- TIME (column) 88, 106, 205
- timer parameter 195
- timesharing mode 196
- timex(1) (command) 30
- TLB, see Translation Lookaside Buffer 16
- TPCLOCK (kernel parameter) 195
- Translation Lookaside Buffer 16, 17, 18, 41, 217
- TRANSVIEW Control Center 57
- truss(1) (command) 30, 89
- TSMAXUPRI (kernel parameter) 195
- TSSLEEPDECAY (kernel parameter) 196
- TSSLEEPMAX (kernel parameter) 196
- TSSLEEPWAIT (kernel parameter) 196
- TSSLPSWPDIV (kernel parameter) 196
- tunable parameter 163, 234
- tunefs(1M) (command) 148
- tunestat(1M) (command) **48**
- tuning 1, **234**
- tuning measures, suitable kernel parameters 165
- type
 - of computer 80
 - of CPU 82
 - of FPU 82
 - of processor 80

- types of kernel parameter 164
 - device driver parameters 194
 - general kernel parameters 183
 - IPC parameters 191
 - message parameters 191
 - parameters for file systems 188
 - parameters for paging 189
 - parameters for STREAMS 189
 - resource limit parameters 196
 - semaphore parameters 192
 - shared memory parameters 193
 - SINIX-specific parameters 198
 - timer and scheduler parameters 195
- U**
 - UFS file system, quota table 188
 - UFSINODE (kernel parameter) 99
 - UFSNINODE (kernel parameter) 12, 26, 90, 149, 168, 173, 188
 - ulimit(1) (command) 167, 196, 197
 - uname(1) (command) 66, 80
 - undo entry 193
 - undo structure 193
 - universe
 - 32-bit address space 11
 - 64-bit address space 11
 - update frequency 56
 - uptime(1) (command) 30, 44, 55
 - user
 - number of processes 184
 - user group 185
 - user ID 181, 184, 187, **235**
 - user interface SYSADM 34–36, 57, **233**
 - user mode 83, 113, 206, 207
 - user segment 10, 18
 - user spin locks 83
 - uucp process 150
- V**
 - V.24 line 202
 - vdisk, see virtual disk
 - version of operating system 80
 - vflt/s (column) 36, 101, 102, 207, 208
 - virtual address space of kernel 10
 - virtual address, field 188
 - virtual channel 202
 - virtual disk 24, 110, 138, 201, 235
 - concatenated disk 25, 139
 - mirror disk 139, 227
 - simple disk 138
 - statesave disk 139
 - striped disk 25, 38, 139, 149
 - virtual main memory 92
 - page **8, 228**
 - virtual memory management 8, 235
 - vm_meas* (procedure) 70, 72
 - vm_mem_eval* (procedure) 70, 92
 - vmstat(1M) (command) 30, 41, 69, **72**, 92, 102
 - d 42
 - f 42
 - m 42
 - S 42
 - s 42
 - VxFS file system 24
 - number of inodes 24, 25
- W**
 - wchar/s (column) 36, 124
 - who(1), command 178
 - write() (system call) 14
 - wsplck (column) 83
- X**
 - X terminal 200
 - X11 client 55
 - xload (command) 33, 55
 - xmtin/s (column) 36

Siemens AG
Information and Communication Products
Computer Systems
User Documentation UNIX
33094 Paderborn
Germany

Fax: (++49) 52518-15248

Internet: DOCetc@pdb.siemens.de
<http://manuals.mchp.siemens.de>

Submitted by

Comments
Suggestions
Corrections

Comments on Reliant UNIX 5.45
Tuning Guide



Siemens Business Services
Training Center, Consulting Service
81730 Munich
Germany

Fax: (++49) 89636-52003

Courses Consulting Self-tuition media

The Siemens Training Center offers consulting for in-service training, courses and self-tuition media on almost all aspects of information technology, either on your premises or at a Training Center near you. Write to us for information and course programs.

Submitted by

send me information/course programs on the following topics



