

*Solaris System
Performance Management*

SA-400

Student Guide



Sun Microsystems
500 Eldorado Boulevard
MS: BRM01-209
Broomfield, Colorado 80021
U.S.A.

Revision B, October 1999

Copyright 1999 Sun Microsystems, Inc., 901 San Antonio Road, Palo Alto, California 94303, U.S.A. All rights reserved. This product or document is protected by copyright and distributed under licenses restricting its use, copying, distribution, and decompilation. No part of this product or document may be reproduced in any form by any means without prior written authorization of Sun and its licensors, if any.

Third-party software, including font technology, is copyrighted and licensed from Sun suppliers.

Parts of the product may be derived from Berkeley BSD systems, licensed from the University of California. UNIX is a registered trademark in the U.S. and other countries, exclusively licensed through X/Open Company, Ltd.

Sun, Sun Microsystems, the Sun Logo, Solaris, NFS, SunOS, Solstice SyMON, HotJava, Solstice DiskSuite, Sun Trunking, SunVTS, OpenWindows, OpenBoot, Ultra Enterprise, Sun StorEdge, SunNet, CacheFS, SunVideo, SunSwift and VideoPix are trademarks or registered trademarks of Sun Microsystems, Inc. in the U.S. and other countries.

All SPARC trademarks are used under license and are trademarks or registered trademarks of SPARC International, Inc. in the U.S. and other countries. Products bearing SPARC trademarks are based upon an architecture developed by Sun Microsystems, Inc.

The OPEN LOOK and Sun Graphical User Interface was developed by Sun Microsystems, Inc. for its users and licensees. Sun acknowledges the pioneering efforts of Xerox in researching and developing the concept of visual or graphical user interfaces for the computer industry. Sun holds a non-exclusive license from Xerox to the Xerox Graphical User Interface, which license also covers Sun's licensees who implement OPEN LOOK GUIs and otherwise comply with Sun's written license agreements.

U.S. Government approval required when exporting the product.

RESTRICTED RIGHTS: Use, duplication, or disclosure by the U.S. Government is subject to restrictions of FAR 52.227-14(g) (2)(6/87) and FAR 52.227-19(6/87), or DFAR 252.227-7015 (b)(6/95) and DFAR 227.7202-3(a).

DOCUMENTATION IS PROVIDED "AS IS" AND ALL EXPRESS OR IMPLIED CONDITIONS, REPRESENTATIONS, AND WARRANTIES, INCLUDING ANY IMPLIED WARRANTY OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE OR NON-INFRINGEMENT, ARE DISCLAIMED, EXCEPT TO THE EXTENT THAT SUCH DISCLAIMERS ARE HELD TO BE LEGALLY INVALID.



Please
Recycle



Contents

About This Course.....	xvii
Course Overview	xviii
Course Map.....	xix
Course Objectives.....	xxvi
Skills Gained by Module.....	xxvii
Guidelines for Module Pacing	xxviii
Topics Not Covered.....	xxix
How Prepared Are You?.....	xxxi
Introductions	xxxii
How to Use the Course Materials.....	xxxiii
Course Icons and Typographical Conventions	xxxv
Icons	xxxv
Typographical Conventions	xxxvi
Introduction to Performance Management.....	1-1
Relevance.....	1-2
What Is Tuning?	1-4
Basic Tuning Procedure	1-5
Conceptual Model of Performance.....	1-6
Where to Tune First	1-8
Where to Tune?	1-9
Performance Measurement Terminology.....	1-10
Performance Graphs.....	1-12
The Tuning Cycle	1-15
SunOS Components.....	1-16
Check Your Progress	1-17
Think Beyond	1-18
System Monitoring Tools.....	2-1
Relevance.....	2-2
How Data Is Collected.....	2-3
The kstat.....	2-4
How the Tools Process the Data	2-5
Monitoring Tools Provided With the Solaris OS.....	2-6

sar	2-7
vmstat	2-8
iostat	2-9
mpstat	2-10
netstat	2-11
nfsstat	2-12
SyMON System Monitor.....	2-13
Other Utilities	2-14
memtool	2-15
/usr/proc/bin.....	2-17
The Process Manager.....	2-18
The SE Toolkit.....	2-20
System Performance Monitoring With the SE Toolkit	2-21
SE Toolkit Example Tools	2-22
System Accounting	2-23
Viewing Tunable Parameters	2-24
Setting Tunable Parameters.....	2-25
/etc/system.....	2-26
Check Your Progress	2-27
Think Beyond	2-28
System Monitoring Tools.....	L2-1
Tasks	L2-2
Using the Tools.....	L2-2
Enabling Accounting.....	L2-3
Installing the SE Toolkit.....	L2-3
Installing memtool.....	L2-4
Processes and Threads	3-1
Relevance.....	3-2
Process Lifetime	3-3
fork With exec Example.....	3-4
Process Performance Issues.....	3-5
Process Lifetime Performance Data.....	3-6
Process-Related Tunable Parameters	3-8
Multithreading	3-10
Application Threads	3-12
Process Execution.....	3-13
Thread Execution	3-14
Process Thread Examples	3-15
Performance Issues	3-16
Thread and Process Performance	3-17
Locking	3-18
Locking Problems	3-20
The lockstat Command	3-21
Interrupt Levels.....	3-22
The clock Routine.....	3-23

Clock Tick Processing.....	3-24
Process Monitoring Using ps	3-25
Process Monitoring Using System Accounting	3-28
msacct.se.....	3-32
pea.se	3-32
ps-ax.se.....	3-34
ps-p.se	3-34
pwatch.se.....	3-34
Check Your Progress	3-35
Think Beyond	3-36
Processes Lab	L3-1
Tasks	L3-2
Using maxusers	L3-2
Using lockstat	L3-3
Monitoring Processes	L3-3
CPU Scheduling	4-1
Relevance.....	4-2
Scheduling States	4-3
Scheduling Classes.....	4-4
Class Characteristics	4-5
Dispatch Priorities.....	4-6
Timesharing/Interactive Dispatch Parameter Table	4-7
CPU Starvation.....	4-8
Dispatch Parameter Table Issues	4-9
Timesharing Dispatch Parameter Table (Batch).....	4-10
The dispadmin Command	4-12
dispadmin Example	4-13
The Interactive Scheduling Class.....	4-14
Real-Time Scheduling	4-15
Real-Time Issues.....	4-16
Real-Time Dispatch Parameter Table.....	4-18
The priocntl Command	4-19
Kernel Thread Dispatching	4-21
Old Thread	4-21
New Thread	4-22
Processor Sets	4-23
Dynamic Reconfiguration Interaction.....	4-24
The Run Queue.....	4-27
CPU Activity.....	4-28
CPU Performance Statistics	4-30
Available CPU Performance Data	4-34
CPU Control and Monitoring.....	4-35
The mpstat Command.....	4-36
mpstat Data.....	4-38
Solaris Resource Manager.....	4-39

SRM Hierarchical Shares Example	4-40
SRM Limit Node	4-41
CPU Monitoring Using SE Toolkit Programs	4-42
cpu_meter.se	4-42
vmmmonitor.se	4-42
mpvmsstat.se	4-42
Check Your Progress	4-43
Think Beyond	4-44
CPU Scheduling Lab	L4-1
Tasks	L4-2
CPU Dispatching.....	L4-2
Process Priorities	L4-3
The Dispatch Parameter Table	L4-6
Real-Time Scheduling	L4-7
System Caches	5-1
Relevance.....	5-2
What Is a Cache?	5-3
Hardware Caches.....	5-4
SRAM and DRAM	5-5
CPU Caches.....	5-6
Cache Operation.....	5-7
Cache Miss Processing	5-8
Cache Replacement.....	5-9
Cache Operation Flow.....	5-10
Cache Hit Rate	5-11
Effects of CPU Cache Misses	5-13
Cache Characteristics.....	5-14
Virtual Address Cache	5-15
Physical Address Cache	5-16
Direct Mapped and Set Associative Caches.....	5-17
Harvard Caches.....	5-18
Write-Through and Write-Back Caches.....	5-19
Write Cancellation	5-19
CPU Cache Snooping	5-21
Cache Thrashing.....	5-23
System Cache Hierarchies	5-25
Relative Access Times	5-27
Cache Performance Issues	5-28
Things You Can Tune	5-30
Check Your Progress	5-31
Think Beyond	5-32
Cache Lab.....	L5-1
Tasks	L5-2
Cache Miss Costs.....	L5-2

Memory Tuning	6-1
Objectives	6-1
Relevance.....	6-2
Virtual Memory.....	6-3
Types of Segments	6-6
Virtual Address Translation.....	6-7
Page Descriptor Cache	6-8
Virtual Address Lookup	6-10
The Memory Free Queue	6-11
The Paging Mechanism.....	6-12
Page Daemon Scanning.....	6-13
Page Scanner Processing.....	6-14
Default Paging Parameters.....	6-15
An Alternate Approach – The Clock Algorithm	6-16
maxpgio	6-18
File System Caching.....	6-20
The Buffer Cache	6-22
The Page Cache	6-24
Priority Paging.....	6-25
Available Paging Statistics.....	6-27
Available Paging Data.....	6-31
Additional Paging Statistics	6-32
Swapping.....	6-34
What Is Not Swappable	6-36
Swapping Priorities	6-37
Swap In.....	6-38
Swap Space	6-39
tmpfs	6-41
Available Swapping Statistics	6-43
Available Swapping Data	6-46
Shared Libraries	6-47
Memory Utilization	6-49
Total Physical Memory	6-50
File Buffering Memory	6-51
Kernel Memory	6-52
Identifying an Application’s Memory Requirements.....	6-54
Free Memory.....	6-56
Identifying a Memory Shortage.....	6-57
Using the memtool GUI.....	6-59
Things You Can Tune.....	6-63
Check Your Progress	6-65
Think Beyond	6-66

Memory Tuning Lab	L6-1
Tasks	L6-2
Process Memory	L6-2
Page Daemon Parameters	L6-3
Scan Rates.....	L6-7
System Buses.....	7-1
Relevance.....	7-2
What Is a Bus?	7-3
General Bus Characteristics.....	7-4
System Buses.....	7-5
MBus and XDbus	7-6
UPA and Gigaplane Bus	7-7
Gigaplane XB Bus.....	7-8
The Gigaplane XB Bus	7-9
Sun System Backplanes Summary.....	7-10
Peripheral Buses.....	7-11
SBus Versus PCI Bus.....	7-12
Enterprise Server I/O Boards.....	7-13
PIO and DVMA.....	7-14
PIO.....	7-15
DVMA.....	7-15
Which Is Being Used?.....	7-15
prtdiag	7-16
prtdiag CPU Section	7-17
prtdiag Memory Section	7-18
Memory Interleaving.....	7-18
prtdiag I/O Section	7-20
Bus Limits.....	7-21
Bus Bandwidth	7-22
Diagnosing Bus Problems.....	7-23
Avoiding the Problem	7-24
SBus Overload Example.....	7-26
Dynamic Reconfiguration Considerations	7-28
Tuning Reports.....	7-29
Check Your Progress	7-30
Think Beyond	7-31
System Buses Lab.....	L7-1
Tasks	L7-2
System Configuration.....	L7-2
I/O Tuning.....	8-1
Relevance.....	8-2
SCSI Bus Overview.....	8-3
SCSI Bus Characteristics	8-5
SCSI Speeds.....	8-6

SCSI Widths	8-8
SCSI Lengths.....	8-9
SCSI Properties Summary.....	8-10
SCSI Interface Addressing.....	8-11
SCSI Bus Addressing.....	8-13
SCSI Target Priorities	8-14
Fibre Channel.....	8-15
Disk I/O Time Components.....	8-18
Access to the I/O Bus	8-18
Bus Transfer Time	8-20
Seek Time	8-20
Rotation Time	8-20
ITR Transfer Time	8-21
Reconnection Time.....	8-21
Interrupt Time	8-21
Disk I/O Time Calculation.....	8-22
Bus Data Transfer Timing.....	8-23
Latency Timing.....	8-23
Disk Drive Features	8-24
Multiple Zone Recording.....	8-25
Drive Caching and Fast Writes	8-27
Fast Writes.....	8-27
Tagged Queueing.....	8-29
Ordered Seeks.....	8-30
Mode Pages.....	8-31
Device Properties From prtconf	8-32
Device Properties From scsiinfo	8-34
I/O Performance Planning	8-36
Decision Support Example (Sequential I/O)	8-37
Transaction Processing Example (Random I/O)	8-37
Tape Drive Operations	8-38
Storage Arrays and JBODs.....	8-39
Storage Array Architecture.....	8-40
RAID	8-41
RAID-0	8-42
RAID-1	8-44
RAID-0+1.....	8-46
RAID-3	8-48
RAID-5	8-50
Tuning the I/O Subsystem	8-52
Available Tuning Data	8-54
Available I/O Statistics	8-55
Available I/O Data	8-58
iostat Service Time.....	8-59
siostat.se	8-60
xio.se	8-60

xiostat.se	8-61
iomonitor.se	8-61
iost.se	8-61
xit.se	8-62
disks.se	8-62
Check Your Progress	8-63
Think Beyond	8-64
I/O Tuning Lab	L8-1
Tasks	L8-2
SCSI Device Characteristics.....	L8-2
UFS Tuning	9-1
Relevance.....	9-2
The vnode Interface	9-3
The Local File System.....	9-5
Fast File System Layout	9-6
Disk Label.....	9-6
Boot Block.....	9-6
Superblock.....	9-7
Cylinder Groups.....	9-7
The Directory	9-9
Directory Name Lookup Cache (DNLC).....	9-10
The inode Cache	9-12
DNLC and inode Cache Management.....	9-14
The inode.....	9-16
inode Time Stamp Updates	9-17
The inode Block Pointers	9-18
UFS Buffer Cache	9-19
Hard Links	9-21
Symbolic Links	9-22
Allocating Directories and inodes	9-23
Allocation Performance.....	9-24
Allocating Data Blocks	9-25
Quadratic Rehash.....	9-26
The Algorithm	9-27
Fragments.....	9-28
Logging File Systems.....	9-29
Application I/O.....	9-30
Working With the segmap Cache	9-33
File System Performance Statistics	9-34
fsflush.....	9-37
Direct I/O.....	9-39
Restrictions.....	9-40
Accessing Data Directly With mmap.....	9-41
Using madvise.....	9-42

File System Performance.....	9-43
Data-Intensive or Attribute-Intensive Applications	9-43
Sequential or Random Accesses	9-44
Cylinder Groups.....	9-45
Sequential Access Workloads.....	9-47
UFS File System Read Ahead	9-49
Setting Cluster Sizes for RAID Volumes	9-51
Commands to Set Cluster Size	9-53
Tuning for Sequential I/O	9-54
File System Write Behind.....	9-55
UFS Write Throttle.....	9-57
Random Access Workloads.....	9-59
Tuning for Random I/O	9-61
Post-Creation Tuning Parameters	9-63
Fragment Optimization.....	9-63
minfree	9-64
maxbpg	9-64
Application I/O Performance Statistics.....	9-65
Check Your Progress	9-67
Think Beyond	9-68
UFS Tuning Lab	L9-1
Tasks	L9-2
File Systems.....	L9-2
Directory Name Lookup Cache	L9-3
mmap and read/write.....	L9-6
Using madvise.....	L9-8
Network Tuning.....	10-1
Relevance.....	10-2
Networks.....	10-3
Network Bandwidth.....	10-4
Full Duplex 100-BaseT Ethernet.....	10-5
ndd Examples.....	10-6
Forcing 100-Base T Full Duplex	10-6
IP Trunking	10-7
Sun Trunking Software	10-8
TCP Connections.....	10-9
TCP Stack Issues.....	10-11
TCP Tuning Parameters.....	10-12
Using ndd	10-14
Network Hardware Performance	10-16
NFS.....	10-18
NFS Retransmissions	10-19
NFS Server Daemon Threads	10-21

Monitoring Network Performance.....	10-23
ping.....	10-24
spray	10-25
snoop	10-26
Isolating Problems	10-27
Tuning Reports.....	10-28
Network Monitoring Using SE Toolkit Programs	10-29
net.se	10-29
netstatx.se	10-29
nx.se	10-29
netmonitor.se.....	10-30
nfsmonitor.se.....	10-30
tcp_monitor.se.....	10-30
Check Your Progress	10-31
Think Beyond	10-32
Performance Tuning Summary	11-1
Relevance.....	11-2
Some General Guidelines.....	11-3
What You Can Do	11-4
Application Source Code Tuning	11-5
Compiler Optimization	11-6
Application Executable Tuning Tips.....	11-7
Performance Bottlenecks.....	11-8
Memory Bottlenecks	11-10
Memory Bottleneck Solutions	11-11
I/O Bottlenecks	11-13
I/O Bottleneck Solutions	11-15
CPU Bottlenecks.....	11-17
CPU Bottleneck Solutions	11-19
Ten Tuning Tips	11-21
Case Study.....	11-23
Check Your Progress	11-46
Think Beyond	11-47
Performance Tuning Summary Lab	L11-1
Tasks	L11-2
vmstat Information	L11-2
sar Analysis I.....	L11-3
sar Analysis II.....	L11-6
sar Analysis III	L11-9
Interface Card Properties.....	A-1
Additional Resources	A-2
SBus Implementations and Capabilities.....	A-3
Typical and Peak Bandwidth of SBus Devices	A-4

Installing and Configuring the SyMON System Monitor.....	B-1
Additional Resources	B-2
SyMON System Monitor.....	B-3
Supported Server Platforms	B-4
Upgrading the SyMON System Monitor.....	B-5
Installing the SyMON System Monitor	B-6
Installing the SyMON Packages	B-7
Environment Variables.....	B-7
Configuring the SyMON System Monitor	B-8
Configuring the Monitored Server System	B-8
Running the SyMON User Interface	B-8
Configuring the Event Handler System	B-9
Setting Up SNMP With Solstice Domain (SunNet)	
Manager.....	B-10
Directing the SNMP Traps.....	B-10
Writing or Modifying Event Rules to Send SNMP	
Traps	B-11
How to Stop or Remove SyMON From the System.....	B-12
Configuration Issues.....	B-13
Accounting	C-1
Overview	C-2
What Is Accounting?	C-2
What Is Accounting Used For?	C-2
Types of Accounting.....	C-3
Connection Accounting.....	C-3
Process Accounting.....	C-3
Disk Accounting.....	C-4
Charging.....	C-4
How Does Accounting Work?	C-5
Location of Files	C-5
Types of Files	C-5
What Happens at Boot Time	C-5
Programs That Are Run	C-6
Location of ASCII Report Files.....	C-7
Starting and Stopping Accounting	C-8
Generating the Data and Reports	C-10
Raw Data	C-10
rprtMMDD Daily Report File.....	C-12
Generation of the rprtMMDD File.....	C-12
The runacct Program.....	C-17
runacct States.....	C-18
Periodic Reports	C-20
Looking at the pacct File With acctcom.....	C-21
acctcom Options.....	C-22
Generating Custom Analyses.....	C-24

Writing Your Own Programs.....	C-24
Raw Data Formats.....	C-24
/var/adm/wtmp File	C-25
/var/adm/pacct File	C-26
/var/adm/acct/nite/disktacct File	C-27
Summary of Accounting Programs and Files.....	C-28
The Cache File System.....	D-1
Introduction	D-2
Function.....	D-2
Characteristics	D-2
Terminology.....	D-2
Benefits	D-3
Limitations	D-4
Required Software	D-4
Setting Up CacheFS	D-5
Creating a Cache	D-5
Displaying Information.....	D-5
Deleting a Cache.....	D-6
Mounting With CacheFS.....	D-7
Using an Already Mounted NFS File Systems	D-7
Using /etc/vfstab.....	D-7
Using the Automounter	D-7
cachefs Operation	D-8
fsck.....	D-8
Writes to Cached Data.....	D-8
Managing cachefs	D-9
cachefsstat	D-9
cachefslog	D-9
cachefswssize.....	D-10
cachefspack	D-10
CacheFS and CD-ROMs.....	D-11
Exercise: Using CacheFS	D-12
Preparation.....	D-12
Tasks	D-13
IPC Tuneables.....	E-1
Additional Resources	E-2
Overview	E-3
How to Determine Current IPC Values.....	E-3
How to Change IPC Parameters	E-4
Tunable Parameter Limits.....	E-5
Shared Memory	E-6
Semaphores.....	E-8
Message Queues.....	E-11

Performance Monitoring Tools	F-1
Additional Resources	F-2
sar	F-3
Display File Access Operation Statistics.....	F-3
Display Buffer Activity Statistics.....	F-3
Display System Call Statistics	F-4
Display Disk Activity Statistics.....	F-6
Display Pageout and Memory Freeing Statistics (in Averages).....	F-7
Display Kernel Memory Allocation (KMA) Statistics	F-8
Display Interprocess Communication Statistics.....	F-9
Display Pagein Statistics	F-9
Display CPU and Swap Queue Statistics	F-10
Display Available Memory Pages and Swap Space Blocks	F-11
Display CPU Utilization (the Default)	F-11
Display Process, Inode, File, and Shared Memory Table Sizes.....	F-12
Display Swapping and Switching Statistics.....	F-12
Display Monitor Terminal Device Statistics	F-13
Display All Statistics.....	F-14
Other Options	F-15
vmstat	F-16
Display Virtual Memory Activity Summarized at Some Optional, User-Supplied Interval (in Seconds).....	F-16
Display Cache Flushing Statistics.....	F-19
Display Interrupts Per Device.....	F-19
Display System Event Information, Counted Since Boot	F-20
Display Swapping Activity Summarized at Some Optional, User-Supplied Interval (in Seconds).....	F-21
iostat	F-22
Display CPU Statistics.....	F-22
Display Disk Data Transferred and Service Times	F-22
Display Disk Operations Statistics	F-23
Display Device Error Summary Statistics	F-23
Display All Device Errors	F-24
Display Counts Rather Than Rates, Where Possible	F-24
Report on Only the First n Disks	F-24
Display Data Throughput in Mbytes/sec Rather Than Kbytes/sec	F-25
Display Device Names as Mount Points or Device Addresses	F-25
Display Per-Partition Statistics as Well as Per-Device Statistics	F-25
Display Per-Partition Statistics Only.....	F-26

Display Characters Read and Written to Terminals	F-26
Display Extended Disk Statistics in Tabular Form	F-27
Break Down Service Time, <code>svc_t</code> , Into Wait and Active Times and Put Device Name at the End of the Line.....	F-28
<code>mpstat</code>	F-29
<code>netstat</code>	F-31
Show the State of All Sockets	F-31
Show All Interfaces Under Dynamic Host Configuration Protocol (DHCP) Control.....	F-34
Show Interface Multicast Group Memberships.....	F-34
Show State of All TCP/IP Interfaces.....	F-35
Show Detailed Kernel Memory Allocation Information..	F-36
Show STREAMS Statistics	F-37
Show Multicast Routing Tables	F-37
Show Multicast Routing Statistics	F-38
Show Network Addresses as Numbers, Not Names.....	F-38
Show the Address Resolution Protocol (ARP) Tables.....	F-39
Show the Routing Tables	F-39
Show Per Protocol Statistics	F-41
Show Extra Socket and Routing Table Information.....	F-42
<code>nfsstat</code>	F-44
Show Client Information Only.....	F-44
Show NFS Mount Options.....	F-46
Show NFS Information; Both Client and Server.....	F-47
Show RPC Information	F-48
Show Server Information only	F-49

About This Course

Course Goal

This course provides an introduction to performance tuning for larger Sun™ environments. It covers most of the major areas of system performance, concentrating on physical memory and input/output (I/O) management.

The Sun environment is quite complex. There are thousands of possible hardware configurations, and tens of thousands of software configurations. Because it is not possible to give “cookbook”-style tuning information, this course examines how the components operate and interoperate. This understanding can be used to determine what kinds of problems can arise, how to identify them, and how to alleviate them.

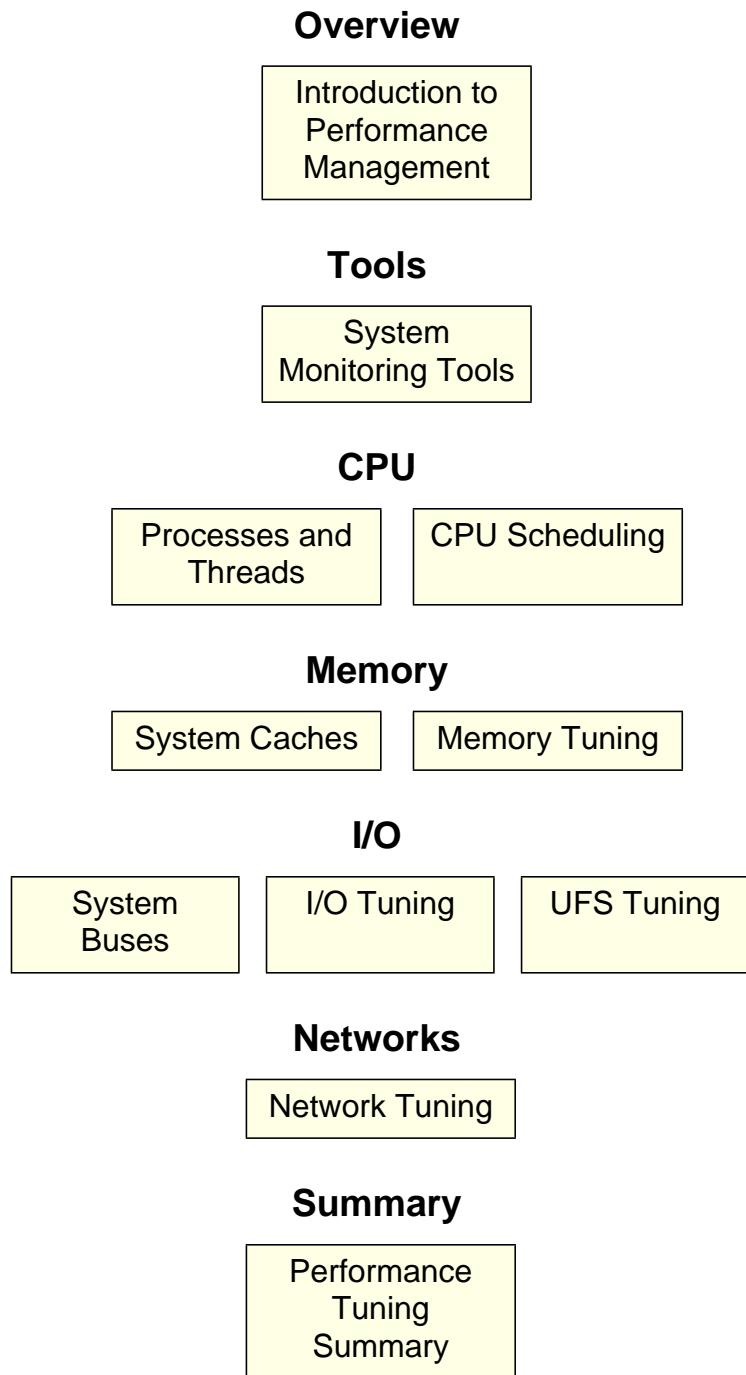
Course Overview

An overview of the operation of various system hardware and software components is given first, then specific areas are examined. This provides an understanding of what effect a certain tuning change will have and whether it is necessary.

This course is intended for system administrators and system designers with a background in Sun SPARC™ system administration.

Course Map

The following course map enables you to see the general topics and the modules for that topic area in reference to the course goal:





Sun Educational Services

Module-by-Module Overview

- Module 1 – "Introduction to Performance Management"
- Module 2 – "System Monitoring Tools"
- Module 3 – "Processes and Threads"
- Module 4 – "CPU Scheduling"
- Module 5 – "System Caches"
- Module 6 – "Memory Tuning"

Module-by-Module Overview

This course contains the following modules:

- Module 1 – “Introduction to Performance Management”

This module introduces the concepts of tuning, tuning tasks, and terminology, and the steps involved in tuning a system. A list of tuning references is also provided.

- Module 2 – “System Monitoring Tools”

The various performance monitoring tools available with the Solaris™ Operating Environment (“Solaris”), the information that they provide, and how to inspect and set system tuning parameters are described in this module.

- **Module 3 – “Processes and Threads”**

This module provides a description of the behavior of processes and threads in the Solaris environment. It covers their creation, execution, and related tuning issues, as well as system timing and the clock thread.

- **Module 4 – “CPU Scheduling”**

How to control work running on the system’s central processing units (CPUs), dispatch priorities, do real-time processing, and use processor sets are all discussed in this module.

- **Module 5 – “System Caches”**

This module describes the generic operation of caches and the specific operation of the hardware caches. Since many system components use caches or operate like one, understanding caches is fundamental to being able to manage system performance.

- **Module 6 – “Memory Tuning”**

The operation of the virtual and real memory subsystems is covered in this module. Because it is perhaps the most important component of system performance, memory operation is discussed in some detail here.



Module-by-Module Overview

- Module 7 – "System Buses"
- Module 8 – "I/O Tuning"
- Module 9 – "UFS Tuning"
- Module 10 – "Network Tuning"
- Module 11 – "Performance Tuning Summary"

- **Module 7 – “System Buses”**

This module shows how buses operate, what kinds of buses there are in the system, and what bus characteristics are important to tuning.

- **Module 8 – “I/O Tuning”**

In this module, the characteristics of the small computer system interface (SCSI) bus and disk drives and their interactions are described. It covers redundant array of inexpensive disks (RAID) levels and storage arrays, and looks at the cost of I/O operations to enable the system administrator to quickly identify and locate the source of an I/O problem.

- **Module 9 – “UFS Tuning”**

This module explains the structure and operation of the Solaris UNIX® file system (UFS). It discusses the various UFS disk structures, caches, and allocation algorithms. A description of the various UFS configuration and tuning parameters is also provided.

- Module 10 – “Network Tuning”

The tuning required for a network, both the physical network and the network applications, is covered in this module. It discusses tuning the Transmission Control Protocol/Internet Protocol (TCP/IP) stack and NFS™.

- Module 11 – “Performance Tuning Summary”

This module summarizes the material covered in this course. It offers examples of poor performance and their causes. It also provides some general guidelines for tuning, a brief discussion of application optimization, and suggestions for dealing with common system performance bottlenecks.



Appendices

- Appendix A – "Interface Card Properties"
- Appendix B – "Installing and Configuring SyMON System Monitor"
- Appendix C – "Accounting"
- Appendix D – "The Cache File System"
- Appendix E – "IPC Tuneables"
- Appendix F – "Performance Monitoring Tools"

Appendices

- Appendix A – “Interface Card Properties”

A summary of the characteristics of many of the I/O interface cards available for the SPARC systems is given in this appendix.

- Appendix B – “Installing and Configuring SyMON System Monitor”

This appendix discusses how to install, configure, and operate the Solaris SyMON™ system monitor.

- Appendix C – “Accounting”

This appendix discusses the configuration and operation of Solaris system accounting.

- Appendix D – “The Cache File System”

The configuration, operation and use of CacheFS™ on both the client and the server is discussed in this appendix.

- Appendix E – “IPC Tuneables”

While not a direct part of the course, the setting of the Interprocess Communication (IPC) shared memory and semaphore parameters is important. This appendix explains all of the IPC tunable parameters.

- Appendix F – “Performance Monitoring Tools”

This appendix is to be used as a reference for the main performance monitoring tools that are part of the Solaris operating system. Sample output and descriptions of the fields are given for the following commands: sar, vmstat, iostat, mpstat, netstat, and nfsstat.

Course Objectives

Upon completion of this course, you should be able to:

- Describe the purpose and goals of tuning
- Describe the common characteristics of system hardware and software components
- Understand how the major system components operate
- Understand the meanings of tuning report elements and how they relate to system performance
- Identify the usual tuning parameters and understand when and how to set them
- Describe the different types of performance graphs
- Determine whether a problem exists in system hardware configurations
- Identify main memory problems and their solutions
- Identify I/O hardware and file system performance problems and their solutions
- Identify network performance problems and their solutions
- Understand CPU scheduling and the functions of threads in the Solaris operating system

Skills Gained by Module

The skills for *Solaris System Performance Management* are shown in column 1 of the following matrix. The black boxes indicate the main coverage for a topic; the gray boxes indicate the topic is briefly discussed.

Skills Gained	Module										
	1	2	3	4	5	6	7	8	9	10	11
Describe the purpose and goals of tuning		■									■
Describe the common characteristics of system hardware and software components					■			■	■	■	
Understand how the major system components operate	■				■			■	■		■
Understand the meanings of tuning report elements and how they relate to system performance			■	■	■	■	■			■	■
Identify the usual tuning parameters and understand when and how to set them			■	■						■	■
Describe the different types of performance graphs		■									■
Determine whether a problem exists in system hardware configurations						■	■	■	■		■
Identify main memory problems and their solutions						■					■
Identify network performance problems and their solutions										■	
Understand CPU scheduling and the functions of threads in the Solaris operating system				■	■						■

Guidelines for Module Pacing

The following table provides a rough estimate of pacing for this course:

Module	Day 1	Day 2	Day 3	Day 4	Day 5
"About This Course"	A.M.				
"Introduction to Performance Management"		A.M.			
"System Monitoring Tools"		P.M.			
"Processes and Threads"		P.M.			
"CPU Scheduling"			A.M.		
"System Caches"			P.M.		
"Memory Tuning"				A.M.	
"System Buses"				P.M.	
"I/O Tuning"					A.M.
"UFS Tuning"					P.M.
"Network Tuning"					A.M.
"Performance Tuning Summary"					P.M.



Topics Not Covered

- Capacity planning
- General system administration
- Network administration and management
- General Solaris problem resolution
- Data center operations
- Disk storage management

Topics Not Covered

This course does not cover the topics shown on the above overhead. Many of these topics are covered in other courses offered by Sun Educational Services.

- Capacity planning – Covered in ES-360: *Planning and Managing A7000 Intelligent Storage Servers*
- General system administration – Covered in SA-237: *Solaris 7 System Administration I*, SA-287: *Solaris 7 System Administration II*, and SA-350: *Solaris 2.x Server Administration*
- Network administration and management – Covered in SA-387: *Solaris TCP/IP Network Administration*
- General Solaris problem resolution – Covered in ST-350: *Sun Systems Fault Analysis Workshop*

-
- Data center operations – Covered in RS-350: *Operating Client-Server Systems*
 - Disk storage management – Covered in SA-347: *Volume Manager With StorEdge A5000*

Refer to the Sun Educational Services catalog for specific information and registration.



How Prepared Are You?

- Experience with Solaris system administration?
- Understand the function of the system kernel?
- Understand the basic organization and operation of the SPARCcenter™ 2000, SPARCserver™ 1000, and Ultra Enterprise 3000, 4000, 5000, and 6000?
- Understand the operation of I/O devices, storage arrays, and the UNIX® file system?
- Have a basic understanding of network protocols?

How Prepared Are You?

To be sure you are prepared to take this course, can you answer yes to the questions shown on the above overhead?

- If you have had Solaris system administration training or experience, you will be able to install software, edit cron tables, start daemons, and perform the lab exercises.
- If you understand the function of the kernel, you will be able to load kernel modules and modify kernel tunable parameters.
- If you have basic knowledge of Sun server platforms, you will understand the role of the CPU, memory, and buses and their relationships to system performance.
- If you have an understanding of system buses, peripheral buses, types of I/O devices, and the UNIX file system, you will be able to recognize inappropriate configurations and tune the I/O subsystem properly for a given application.
- If you understand network protocols, you will be able to tune them in order to maximize the network bandwidth.



Introductions

- Name
- Company affiliation
- Title, function, and job responsibility
- Performance tuning experience
- Experience with performance monitoring tools
- Reasons for enrolling in this course
- Expectations for this course

Introductions

Now that you have been introduced to the course, introduce yourself to each other and the instructor, addressing the items shown on the above overhead.



How to Use the Course Materials

- Objectives
- Relevance
- Overhead image
- Lecture
- Exercise
- Check Your Progress
- Think Beyond

How to Use the Course Materials

To enable you to succeed in this course, these course materials employ a learning model that is composed of the following components:

- **Course map** – An overview of the course content appears in the "About This Course" module so you can see how each module fits into the overall course goal.
- **Objectives** - What you should be able to accomplish after completing this module is listed here.
- **Relevance** – This section, which appears in every module, provides scenarios or questions that introduce you to the information contained in the module and provoke you to think about how the module content relates to optimizing system performance.
- **Additional resources** – Where you can look for more detailed information, or information on configurations, options, or other capabilities on the topics covered in the module or appendix.

-
- **Overhead image** – Reduced overhead images for the course are included in the course materials to help you easily follow where the instructor is at any point in time. Overheads do not appear on every page.
 - **Lecture** – The instructor will present information specific to the topic of the module. This information will help you learn the knowledge and skills necessary to succeed with the exercises.
 - **Exercise** – Lab exercises will give you the opportunity to practice your skills and apply the concepts presented in the lecture.
 - **Check your progress** – Module objectives are restated, sometimes in question format, so that before moving on to the next module you are sure that you can accomplish the objectives of the current module.
 - **Think beyond** – Thought-provoking questions are posed to help you apply the content of the module or predict the content in the next module.

Course Icons and Typographical Conventions

The following icons and typographical conventions are used in this course to represent various training elements and alternative learning resources.

Icons



Additional resources – Indicates additional reference materials are available.



Discussion – Indicates a small-group or class discussion on the current topic is recommended at this time.

Note – Additional important, reinforcing, interesting, or special information.



Caution – A potential hazard to data or machinery.



Warning – Anything that poses personal danger or irreversible damage to data or the operating system.

Typographical Conventions

Courier is used for the names of commands, files, and directories, as well as on-screen computer output. For example:

Use `ls -al` to list all files.
system% You have mail.

Courier **bold** is used for characters and numbers that you type. For example:

system% **su**
Password:

Courier italic is used for variables and command-line placeholders that are replaced with a real name or value. For example:

To delete a file, type `rm filename`.

Palatino italics is used for book titles, new words or terms, or words that are emphasized. For example:

Read Chapter 6 in *User's Guide*.
These are called *class* options.
You *must* be root to do this.

Introduction to Performance Management

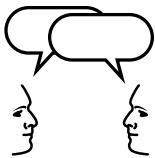
1 

Objectives

Upon completion of this module, you should be able to:

- Explain what performance tuning really means
- Characterize the type of work performed on a system to make meaningful measurements
- Explain performance measurement terminology
- Describe a typical performance tuning task

Relevance



Discussion – The following questions are relevant to understanding the content of this module:

- Why tune a system?
- How do you know that a system requires tuning?
- How do you know what to tune?

Additional Resources



Additional resources – The following references can provide additional details on the topics discussed in this module:

- Cockcroft, Adrian, and Richard Pettit. 1998. *Sun Performance and Tuning*. Palo Alto: Sun Microsystems Press/Prentice Hall.
- Wong, Brian L. 1997. *Configuration and Capacity Planning for Solaris Servers*. Palo Alto: Sun Microsystems Press/Prentice Hall.
- Benner, Alan F. 1996. *Fibre Channel*. McGraw-Hill.
- Catanzaro, Ben. 1994. *Multiprocessor System Architectures*. SunSoft Press/Prentice Hall.
- Cervone, Frank. 1998. *Solaris Performance Administration*. McGraw-Hill.
- Goodheart, Berny, and James Cox. 1994. *The Magic Garden Explained*. Prentice-Hall.
- Graham, John. 1995. *Solaris 2.x Internals*. McGraw-Hill.
- Gunther, Neil. 1998. *The Practical Performance Analyst*. McGraw-Hill.
- Raj Jain, Raj. 1991. *The Art of Computer System Performance Analysis*. John Wiley & Sons, Inc.
- Ridge, Peter M., and David Deming. 1995. *The Book of SCSI*. No Starch Press.
- *Solaris 2 Security, Performance and Accounting in the Solaris System Administration AnswerBook™*.

-
- *SMCC NFS Server Performance Tuning Guide* in the Hardware AnswerBook on the SMCC Supplements CD-ROM.
 - The Sun web site <http://www.sun.com/sun-on-net/performance.html> and related pages.
 - Man pages for the various commands (kstat, ps, lockstat, prtconf, prtdiag, sysdef, adb, ndd, fstyp, mkfs, newfs, tunefs) and performance tools (sar, vmstat, iostat, mpstat, nfsstat ,and netstat).



Sun Educational Services

What Is Tuning?

- Improve resource usage
- Improve user perception of the system
- Make more efficient use of system resources
- Increase system capacity
- Adapt the system to your workload
- Lower costs

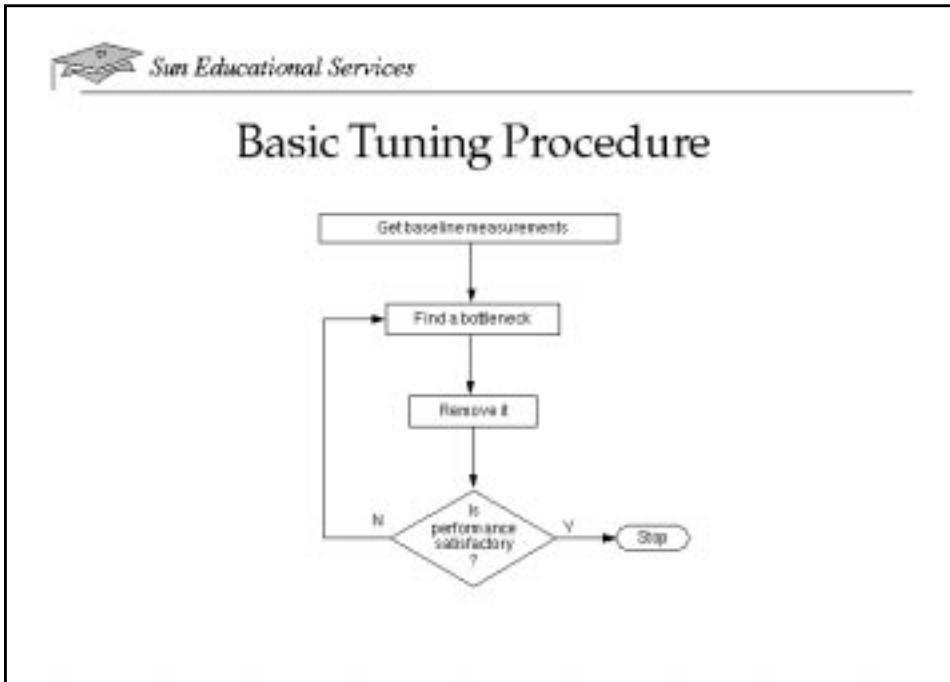
What Is Tuning?

As shown in the overhead image above, tuning means many things. In general, however, it means making the most efficient use of the resources that the system has for the workload that runs on it.

Tuning is aimed at two primary activities:

- Eliminating unnecessary work performed by the system
- Taking advantage of system options to tailor the system to its workload

Almost every tuning activity can be placed into one of these two categories.

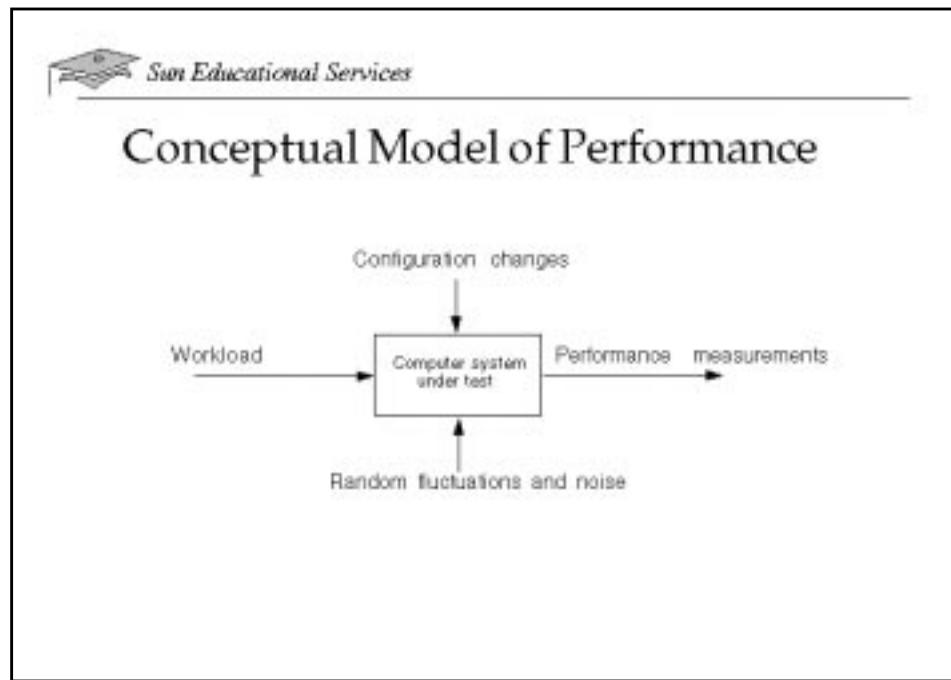


Basic Tuning Procedure

Basic tuning procedures are quite simple: locate a bottleneck and remove it. This process is repeated until performance is “satisfactory,” however that is measured.

Note that you will never be finished tuning: there will always be a bottleneck. If there were none, infinite work could be completed in zero time. Tuning provides a compromise between cost, the adding of extra hardware, and performance, the ability to get the system’s work done efficiently.

Before beginning a tuning project, you should have a goal in mind: what is satisfactory performance. A realistic goal is important: expectations can be set that cannot be met, causing even the most successful tuning project to be perceived as a failure.



Conceptual Model of Performance

There are many factors that make a system perform the way it does. These include:

- The workload itself. The workload may be I/O bound, CPU bound, or network intensive, consisting of short jobs; long jobs; simple queries; long, complex queries; simple text editing; complex computer-aided design (CAD) simulations; or any combination of these. It may change during the day or week.
- A random component to the work; that is, how many queries are executed, how many records are looked at, how many users are supported, how much email is generated, how big files are, what the file locations are, and so on. The exact same amount and type of work is rarely done repeatedly on a system.
- Changes to the system itself. These include new hardware components, new network access, new software, new algorithms from patches, or different media.

All of these items, and more, work together to make the system a very dynamic entity, constantly changing and acting differently over time.

When tuning, you need as much information as possible to make correct decisions about the system's current and future behavior, and where efficiencies can be obtained.

 Sun Educational Services

Where to Tune First

Priority	Area	Example
	OS parameters	<code>nosize</code>
	DBMS	Shared memory
	Application	File access pattern
	Server and network hardware	Disk layout

Where to Tune First

Take a moment to prioritize the areas in the overhead image for tuning purposes from 1 (most improvement possible) to 4 (least improvement).



Where to Tune?

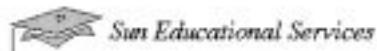
- The closer to the application you tune, the more leverage you have.
- System administrators have little control at the application or DBMS levels.
- For the OS and hardware, general tuning often benefits all applications.
 - You can tune the system specifically for one application if you accept the trade-offs.
 - You need to understand the applications' characteristics to properly tune them.

Where to Tune?

As shown on page -8, the closer to where the work is performed, the more effect your tuning will have. As system administrators, access to the first two levels is difficult, so the tendency is to concentrate on what is accessible.

Tuning at the lower levels is usually an attempt to make the system support the applications as efficiently as possible. This usually involves making trade-offs. Changes that benefit one type of work may have no effect on, or could hinder, another type of work. If you tune the system specifically for a particular type of work, you must be prepared to accept these trade-offs.

You also need to understand the characteristics of the applications to know how they use the system before you can tune the system properly to support them.



Performance Measurement Terminology

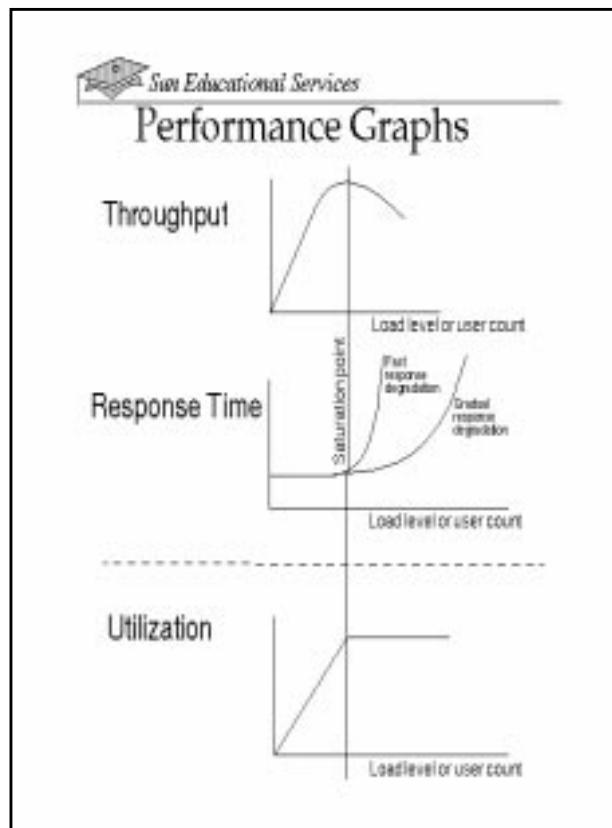
- Bandwidth
- Throughput
- Response time
- Service time
- Utilization

Performance Measurement Terminology

The terminology used in the tuning literature is:

- *Bandwidth*
 - ▼ The peak capacity that cannot be exceeded
 - ▼ A measurement that ignores overhead
 - ▼ The best (perfect) case numbers
 - ▼ Something that is never achieved in practice; there are always some inefficiency and overhead

- *Throughput*
 - ▼ How much work gets done in a specified time interval
 - ▼ What you really get; that is, how much of the bandwidth is actually used
 - ▼ A measurement that is dependent on many factors, including hardware, software, human, and random
 - ▼ A number that is usually impossible to calculate, but can be approximated
 - ▼ Maximum throughput if you are measuring how much work can be done at 100 percent utilization
- *Response time*
 - ▼ How long the user or requester waits for a request to complete
 - ▼ Elapsed time for an operation to complete
 - ▼ Usually the same thing as latency
- *Service time*
 - ▼ Actual request processing time
 - ▼ How long a request takes to perform, ignoring any queueing delays
 - ▼ Response time, which is the same as service time when there is no queueing
- *Utilization*
 - ▼ How much of a resource, either aggregate or individual, was used to do the measured work
 - ▼ The amount of throughput accomplished by a resource
 - ▼ A percentage usually expressed as a busy percentage



Performance Graphs

The three graphs on this page show the most common appearance of data obtained from performance measurements.

The first two graphs indicate a problem, but not its source. You can see that the work performed, measured either as throughput or response time, improves to a certain level, and then begins to drop off. The drop-off may occur quickly or slowly, but the system is no longer able to efficiently perform more work. A resource constraint has been reached.

The third graph, a utilization graph, shows how much capacity is available for a given resource. When fully used, the resource is no longer available and requesters have to wait for it.

When there is a drop-off, as measured by the first two types of graph, there will be a resource that is fully utilized, as measured by the third graph. Locating this resource and increasing its availability is the goal of tuning.



Sample Tuning Task

- Confirm what the problem appears to be
- Collect appropriate tuning data
- Locate the problems
 - Generic system problems
 - Application-specific problems
- Develop a plan to fix them
- Fix what is reasonable
 - Do not fix what is not broken
- Verify performance is acceptable

Sample Tuning Task

Most tuning tasks begin with the complaint, “The system’s too slow.”

This is a very general statement, and means different things to different people. Before beginning a tuning project, you must identify the problem: precisely *what* is too slow. It may be that there is no problem, the user has unrealistic expectations, or there really is some kind of a problem.

Once you have confirmed the problem, you must gather relevant system performance information. It may not be possible to quickly identify the source of the problem. In this case, general information must be gathered and analyzed to locate likely causes.

Once this has been done, various solutions must be evaluated and chosen for implementation. Some may be as simple as changing a tuning parameter, some may require recompiling the application, and others may require the purchase of additional hardware.

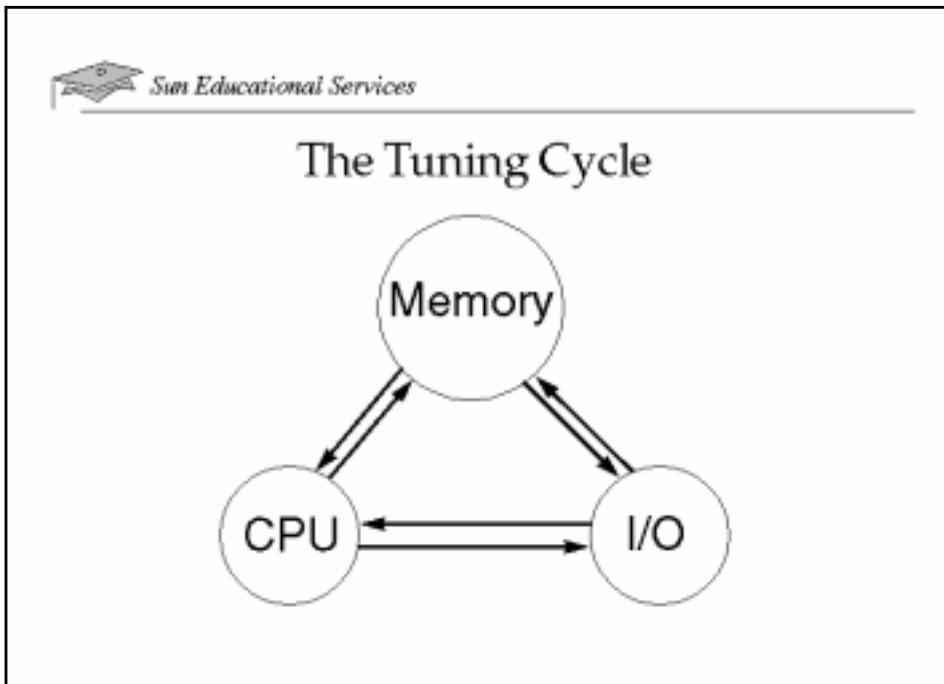
The more that you know about the system and its applications, the more options you will have for performance improvements, and the more cost efficient you can be. Spending money on a new system, software, or components may be the easiest solution, but it is not always the best.

Once you have identified the source of the problem, develop and implement a plan to remove the cause. Make sure that you only fix what is reasonable to fix; requests to software vendors to redesign their product are usually futile.

If you are not sure what a parameter does, do not change it or be prepared to change it back if it damages your performance. Be very careful; some parameters can affect system integrity, and you should not change them without understanding what they do.

You should also make as few changes to the system at a time as possible. Reevaluate performance after each change. Some changes may have a negative impact, which may be masked if multiple changes are made at the same time. By making a few changes at a time, you can quickly identify ineffective changes and gather more information on the effects of certain changes.

Finally, after a change, make sure that you continue to monitor the system. You need to ensure that performance is now “satisfactory,” as well as check to see that a new bottleneck (and there will always be one) is not causing new problems in another part of the system.

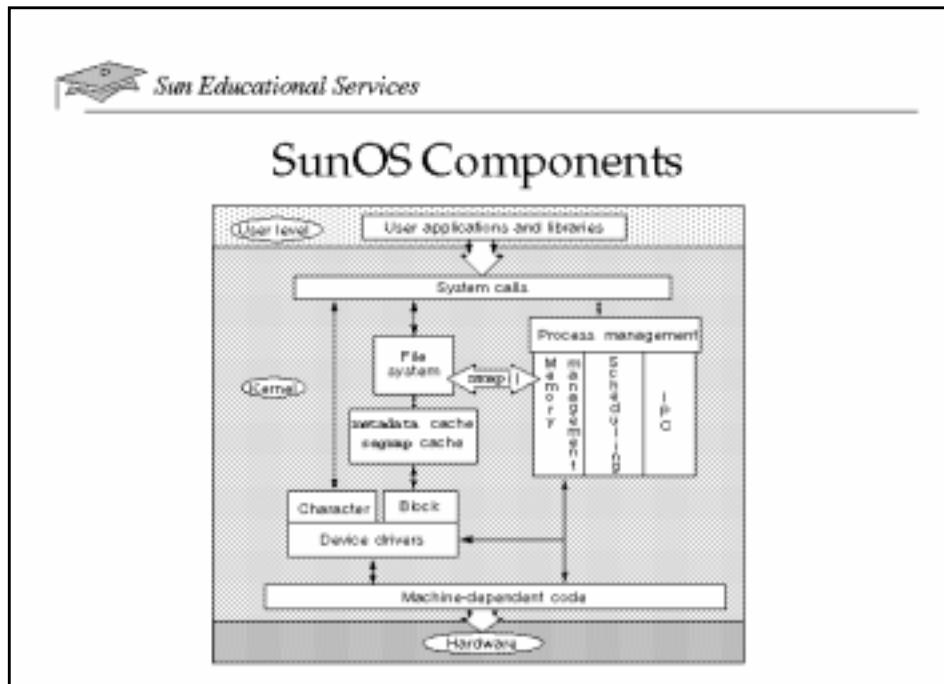


The Tuning Cycle

It has been said that tuning is like peeling an onion, there is always another layer and more tears. As shown in the above overhead, the tuning cycle is continuous.

As discussed earlier, there will always be a bottleneck. When one bottleneck, or utilization problem, has been relieved, another will take its place.

For example, fixing a CPU problem may reveal (or create) an I/O problem. Fixing that may reveal a new CPU problem, or perhaps a memory problem. You always move from one bottleneck to another, until the system is (for the moment) performing acceptably.



SunOS Components

The diagram in the overhead image shows the significant components of the Sun SunOS™ operating system and some of their interactions.

Many of these components will be examined during the course, to see how problems with them can be identified and eliminated.

There will also be an in-depth look at some aspects of the hardware, since there are many similarities in the operation of the hardware and the software components. Understanding one can help in understanding and working with the others.

Check Your Progress

Before continuing on to the next module, check that you are able to accomplish or answer the following:

- Explain what performance tuning really means
- Characterize the type of work performed on a system to make meaningful measurements
- Explain performance measurement terminology
- Describe a typical performance tuning task

Think Beyond

Ask yourself the following questions:

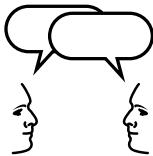
- Are there areas that will never cause a bottleneck?
- Why bother to tune a system at all?
- Are default system parameters geared toward desktop systems or servers? Why?
- How long should a tuning project take? How do you know?
- How can you determine a realistic and “satisfactory” performance goal ahead of time?

Objectives

Upon completion of this module, you should be able to:

- Describe how performance data is collected in the system
- Describe how the monitoring tools retrieve the data
- List the data collection tools provided by the Solaris environment
- Explain how to view and set tuning parameters
- Describe the system accounting functions
- Describe the use of the SolsticeTM SyMONTM system monitor

Relevance



Discussion – The following questions are relevant to understanding the content of this module:

- What monitoring tools are provided with the Solaris operating system?
- Where do the tools get the information that they report?
- When do you use each of the tools?
- Why do the monitoring tools overlap in what they report?

Additional Resources



Additional resources – The following references can provide additional details on the topics discussed in this module:

- Cockcroft, Adrian, and Richard Pettit. 1998. *Sun Performance and Tuning*. Palo Alto: Sun Microsystems Press/Prentice Hall.
- Wong, Brian L. 1997. *Configuration and Capacity Planning for Solaris Servers*. Palo Alto: Sun Microsystems Press/Prentice Hall.
- Man pages for the various commands (`kstat`, `sysdef`, `adb`, `ndd`) and performance tools (`sar`, `vmstat`, `iostat`, `mpstat`, `nfsstat`, `netstat`).
- Mauro/McDougall. 1999. *Solaris Architecture and Internals*. Sun Microsystems Press/Prentice Hall.



Sun Educational Services

How Data Is Collected

- Have the data collection tools read counters and other data structures
- Wait for the specified interval
- Reread the data structures
- Calculate and scale the counter differences
- Remember, the first reading uses zeros
 - Ignore the first line in monitoring reports

If the kernel routines do not collect the data, it cannot be reported.

How Data Is Collected

Performance monitors report two types of data: samples and counts. The data that they collect is usually not gathered directly by the monitor, but by the various operating system components.

At regular intervals, the monitors access the component data structures and save the measurement information. Depending on the information, the previous measurement may be subtracted from the current to identify the activity during the interval.

The first output line reported by the Solaris tools provides the current values of the counters. This first line should be ignored because this is not reliable summary data. Subsequent lines are accurate and subtracted from the previous values.

There is no way for the monitors to collect information. Therefore, if the kernel routines do not collect it, the information cannot be reported.



The kstat

- Is created by a system component
- Exists for every thing the component wants to measure
- Has a specific format for each component
- Is read by the kstat-related system calls
 - A copy is made into the requesting process.
- Allows whole chain of kstats to be read
- Is updated by a component as needed
 - Usually, updates are made when the recorded event occurs.

The kstat

The kstat is a generic kernel data structure that is used by system components to report measurement data. The system may have dozens or even hundreds of kstat structures, with various formats for the different components and reported data.

The kstats all have names, and can be read either by name or by following a chain. There are a series of kstat manipulation system calls (such as `kstat_lookup`); they are all documented in the man pages.

The owning system component is responsible for keeping the contents of the kstat current, usually by incrementing the appropriate counters as the measured event occurs.



Sun Educational Services

How the Tools Process the Data

- Data is accumulated during each interval.
- The change from the previous interval is calculated.
- The information is written:
 - To the screen (stdout)
 - To a graph (such as by SyMON™ system monitor)
 - To a file for post processing
- Post processing can combine intervals.
 - Be careful of too much smoothing

How the Tools Process the Data

As mentioned earlier, the tools accumulate data over each measurement interval and are scaled as appropriate. The difference between the current interval and the previous interval is calculated. Once this has been done for all of the data elements being recorded, the data is sent to the specified destination, such as a file.

When processing this data, you can specify intervals that are multiples of the tool's internal recording interval. Be careful when specifying a large interval, spikes and other unusual readings can be smoothed out and lost, causing you to see no problems. But when shorter intervals are examined, problems become visible.

Remember that intervals reported by the different tools, even if they are of the same length, will start at different times, making it difficult to coordinate events between reports.



Tools Provided With the Solaris OS

- sar
- vmstat
- iostat
- mpstat
- netstat
- nfsstat
- SyMON system monitor
- Other utilities

Monitoring Tools Provided With the Solaris OS

The Solaris operating system (OS) provides a number of monitoring tools. Each of these tools are briefly introduced over the next several pages. Usage of the monitoring tools and specific options and meanings of the data reported by each tool are integrated in the remaining modules where applicable. Other tools are presented in subsequent modules as well.

When more than one tool reports similar data, a table comparing the data produced by the different tools will be provided.



sar

- Does general data collection
- Has 16 different collection types
- Covers:
 - File system, and system calls
 - Physical memory usage, paging, swap, and kernel memory
 - Block device activity and TTY activity
 - Process table status
 - SV IPC activity and CPU usage

sar

The system activity reporter, `sar`, provides a wide range of measurement information covering most areas of the system.

There are 16 different reports, which can be combined, that collect large quantities of simultaneously reported data. The `sar` command with no options provides CPU activity information.

The `sar` data is usually recorded to a file, then processed later due to the large amount of available data.

 Sun Educational Services

vmstat

- Reports primarily processor-related statistics
- Has five reports
- Covers:
 - Process, interrupt, CPU cache, and CPU activity
 - Virtual memory and swapping activity
 - Disk activity
- Provides information similar to `sar`

vmstat

The `vmstat` (virtual memory statistics) report focuses on memory and CPU activity. It has five different reports, including one report on limited disk activity numbers.

The information that `vmstat` reports is similar to that reported by `sar`, but in a few cases, reports data not provided by `sar`.

Averaged rates of kernel counters are displayed, which are calculated as the difference between two snapshots of a number. The first line of output is the difference between the current value of the counters and zero.

As will be discussed later, paging I/O has several distinct uses. These are reported by the `memstat` tool, which is merged into `vmstat` in the Solaris 7 OS.



iostat

- Reports I/O device performance data
- Includes NFS™ mount points and raw devices
- Has 13 operands
- Covers:
 - CPU utilization and TTY activity
 - Disk activity and device errors
- Uses -M to display Mbytes per second instead of Kbytes
- Uses -n to display cXtXdX device names

iostat

The **iostat** (I/O statistics) command reports I/O performance data for disk and terminal (TTY) devices. It also provides CPU utilization information. **iostat** has been significantly extended for the Solaris 2.6 release; it now provides the ability to track NFS mount points, and partitions, and report device error rates.

In addition, a new **-n** option has been provided to make the identification of individual mount points more recognizable. The **-p** option is used to report data at the individual partition level.

If you are using the Volume Manager, **iostat** may not show you enough information about the volumes. You will need to use the **vxstat** command, which is provided with Volume Manager.



mpstat

- Reports CPU-related activity
- Reports CPU-specific statistics
- Has one report
- Covers:
 - Page faults, interrupts, and cross-calls
 - Locking context switches, and CPU utilization
 - Is usually not needed

mpstat

The `mpstat` (multiprocessor statistics) command provides activity information for individual CPUs. It reports CPU utilization information and gives the frequency of occurrence for events such as interrupts, page faults, and locking.

The data from `mpstat` are not usually needed during routine tuning, but can be very helpful if there is unusual behavior or the other monitoring tools show no indication of the source of a problem.

Output from the `mpstat` report displays one line for each CPU in the system.

At this time, `mpstat` does not provide information on processor set activity.



netstat

- Reports network-related activity
- Has 15 different reports
- Covers:
 - Socket state, interface state, and DHCP information
 - arp and routing tables
 - STREAMS statistics
- Gives you excellent visibility on the use of your network
- Can be complicated if you have many network interfaces

netstat

The netstat (network statistics) command reports network utilization information, including socket usage, Dynamic Host Configuration Protocol (DHCP) information, arp information, and traffic activity.

While netstat information can be critical to tuning your network, it can also be difficult to interpret for a system with many network interfaces.

The default report provides active socket and window size information.

The **-i** option is used to show the number of packets transmitted and received by each interface.

The **-s** option shows extremely detailed counts for very low-level TCP, User Datagram Protocol (UDP), and other protocol messages. This allows for quick identification of problems or unusual usage patterns.



Sun Educational Services

nfsstat

- Reports NFS-related activity
- Has five reports
- Covers:
 - Client-side information
 - NFS-mounted file system information
 - NFS and RPC information
- Can help isolate the exact source of an NFS problem
- Can be useful with DBMS network problems as well
 - Use RPCs to transfer requests and data.

nfsstat

The **nfsstat** (NFS statistics) command has six options that provide detailed information on a system's NFS activity. It provides both host and client information, which can be used to locate the source of a problem. Also, since remote procedure call (RPC) information is provided, it can also report on the network activity of some database management systems (DBMSs).

nfsstat can be used to reinitialize statistical information about NFS and RPC interfaces to the kernel. If no options are given, the default command is **nfsstat -cnrs**, which displays everything and reinitializes nothing.

Without options, **nfsstat** provides a report which lists the various types of messages received by NFS. Every NFS message type is reported.



SyMON System Monitor

- Is provided on the SMCC Supplements CD-ROM
- Provides remote monitoring of:
 - Performance data
 - System failures
 - System configuration
 - Syslog information
- Uses Simple Network Management Protocol (SNMP) to transfer data
- Uses rules to signal exception conditions

SyMON System Monitor

The SyMON system monitor on the SMCC Supplements CD-ROM, is shipped with Sun's Solaris server kit. (The latest version can be downloaded from the Sun web site.) The SyMON system monitor is available on all midrange servers (Enterprise 3000, 3500, 4000, 4500, 5000, 5500, 6000, and 6500), and the high-end server (Enterprise 10000 [E10000]).

It provides for the remote monitoring of system configuration, performance, and failure notification. It uses Simple Network Management Protocol (SNMP) to receive much of the data that it reports.

SyMON exception reporting is managed by rules, which can be changed and added to by the user. The user can add monitoring and notification of exception conditions specific to an individual site or system.

The installation and usage of the SyMON system monitor is documented in Appendix B.



Other Utilities

- memtool
- /usr/proc/bin
- Process Manager
- The SE Toolkit
- System accounting

Other Utilities

There are several other utilities and performance monitors provided by Sun that can be useful in identifying performance problems. Each of the following will be discussed in this module:

- memtool
- /usr/proc/bin
- Process Manager
- The SE Toolkit
- System accounting



Sun Educational Services

memtool

- Is not provided with the Solaris OS
 - Retrieve latest version by sending a request to memtool-request@chessie.eng.sun.com or <ftp://playground.sun.com/pub/memtool>
 - Shows detailed information on memory usage
 - Reports on physical memory and swap
 - Provides real-time reports on:
 - Process memory, file memory, and shared library usage

memtool

memtool was developed with the intent of providing a more in-depth look at memory allocation on a Solaris system. It is not included with the Solaris 7 operating system. Earlier versions were named bunyip.

It provides real-time graphic reports on the current memory usage by processes, files, and shared libraries. The displayed information can be filtered by different criteria.

Several command-line, character, and graphical user interface (GUI) tools are provided with the `memtool` package. Table 2-1 lists the tools.

Table 2-1 `memtool` Utilities

Tool	Interface	Description
<code>pmem</code>	Command-line	Processes memory map and usage
<code>memps</code>	Command-line	Dumps process summary and UFS
<code>memtool</code>	GUI	Provides comprehensive GUI for UFS and process memory
<code>mem</code>	Curses user interface (CUI)	Provides a curses interface for UFS and process memory



/usr/proc/bin

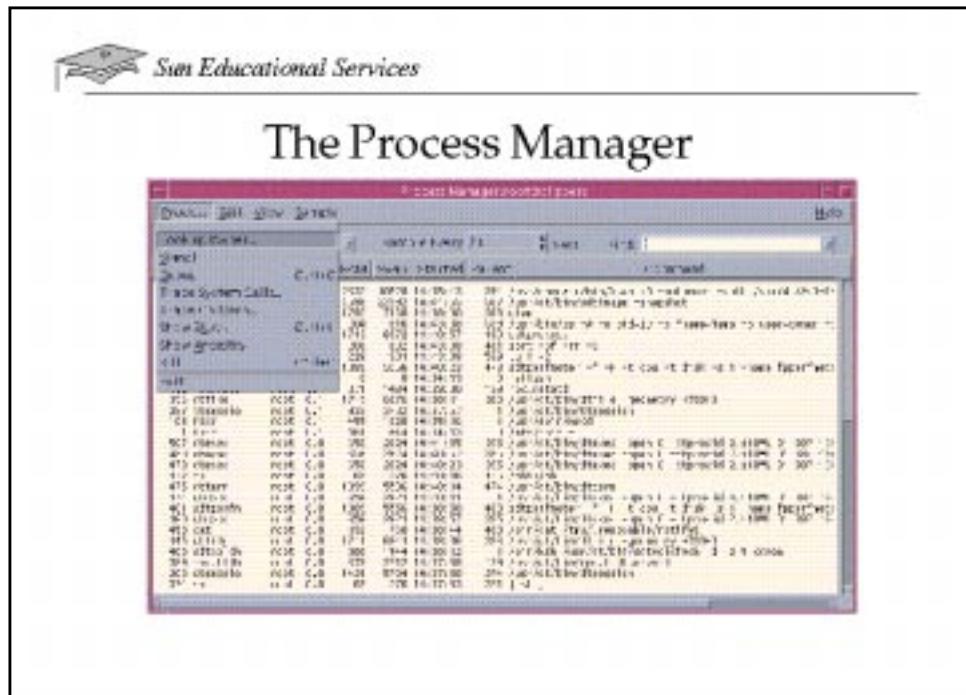
- This directory was added to the Solaris 2.5.1 operating system.
- It contains 13 commands that format data from /proc.
- Commands include:
 - ptree – Display process' ancestors and children
 - pmap – Display process' virtual memory segments
 - pfiles – Open file information for a process
- The output can be cryptic.

/usr/proc/bin

The Solaris 2.5.1 OS added a new set of utilities to the /usr/proc/bin directory. These utilities interface with the /proc file system, allowing detailed process information to be reported.

Among the commands provided are ptree, which prints an indented listing showing the relationship of a process' ancestors and children; and pfiles, which provides some (rather cryptic) information on the files used by a process.

All of the /usr/proc/bin commands are documented on the proc man page.



The Process Manager

The Process Manager can be used to quickly locate a problem process and track current system resource usage. It is installed with the Solaris Desktop Extension (SDX), also known as SolarEZ, which is available on the SMCC Supplements CD-ROM with the Solaris 2.6 3/98 and later releases OS. It is a standard part of Solaris 7.

The SolarEZ's common desktop environment (CDE) Front Panel and menu items are different than the Solaris 7's. The Process Manager is started through the Tools submenu.

The Process Manager displays data retrieved from the `ps` command and the `/proc` file system. It will display:

- CPU usage
- I/O usage
- Memory usage
- Process credentials
- Tracebacks and traces
- Scheduling parameters

It can invoke several of the `/usr/proc/bin` commands for a process, as well as `truss` and `kill`. Process information can be sorted by any available category.

The Process Manager replaces `proctool`, which is a utility that has been provided for a long time as an unsupported service by the Sun Enterprise Services organization.



The SE Toolkit

- Has three packages that provide modifiable tools to monitor your system
- Provides graphic current and historical performance information
- Will offer suggestions on avoiding or resolving problems that it detects
- Looks at disks, networks, NFS, swap space, memory, CPUs, mutexes, and other areas
- Is available from <http://www.sun.com/sun-on-net/performance/se3>

The SE Toolkit

The SE Toolkit is a customized C interpreter which contains a large number of scripts. The scripts are written in the SE language—which is also known as SymbEL—and provide lots of examples of how to write simple performance tools.

The SE Toolkit consists of three packages, RICHPse, RICHPsex, and ANCrules, which together provide a set of easily modifiable performance monitoring tools. These packages, including additional patches and updates, are available from the <http://www.sun.com/sun-on-net/performance/se3> web site.

Once installed, the tools can be configured to offer notices and repair suggestions as it detects problems. If run by root, one of the utilities (`virtual_adrian.se`) will actually change some of the system tuning parameters as conditions require.

Remember that it may be necessary to make changes to the default rules provided with the SE Toolkit to adapt it to your system configuration and workload.

System Performance Monitoring With the SE Toolkit

One of the SE programs is a GUI front end which displays the state of your system. The program is called `zoom.se` and a typical display is shown in .

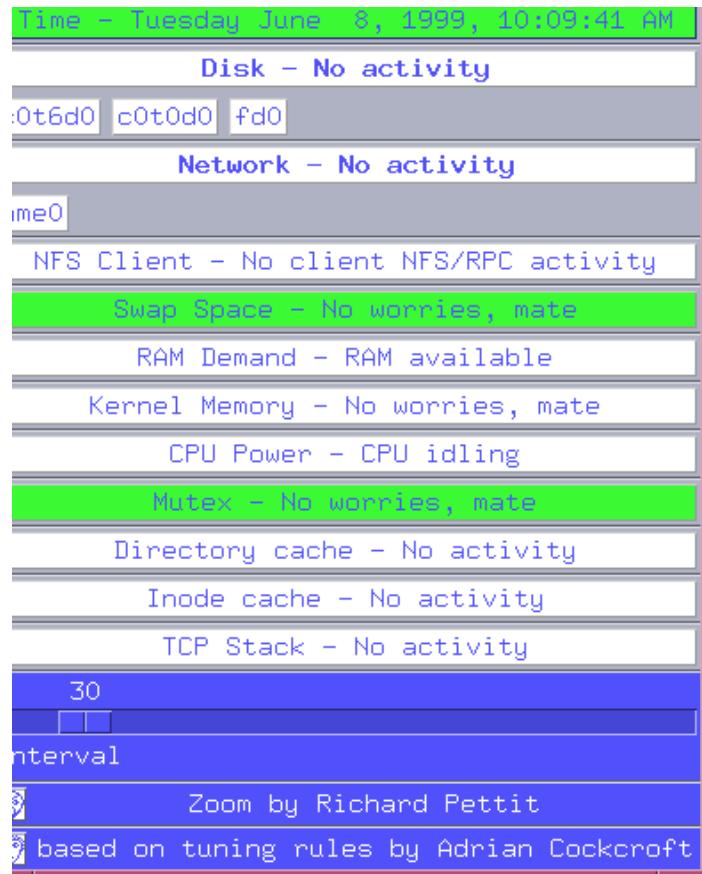


Figure 2-1 Example Display of `zoom.se`

The `zoom.se` monitor displays the status of the main components of a system using color-coded states and action strings. The status colors and action strings are based on a set of performance rules that are coded into the SE Toolkit.

SE Toolkit Example Tools

The SE Toolkit has several tools. Source code for each of the tools is provided. Table 2-2 lists some of the example tools by function.

Table 2-2 SE Programs Grouped by Function

Function	Example SE Programs		
Rule monitors	cpg.se	monlog.se	mon_cm.se
	live_test.se	percollator.se	zoom.se
	virtual_adriาน.se	virtual_adrian_lite.se	
Disk monitors	siostat.se	xio.se	xiostat.se
	iomonitor.se	iost.se	xit.se
	disks.se		
CPU monitors	cpu_meter.se	vmmonitor.se	mpvmsstat.se
Process monitors	msacct.se	pea.se	ps-ax.se
	ps-p.se	pwatch.se	
Network monitors	net.se	tcp_monitor.se	netmonitor.se
	netstatx.se	nx.se	nfsmonitor.se
Clones	iostat.se	uname.se	vmstat.se
	nfsstat.se	perfmonitor.se	xload.se
Data browsers	aw.se	infotool.se	multi_meter.se
Contributed code	anasa	dfstats	kview
	systune	watch	
Test programs	syslog.se	cpus.se	pure_test.se
	collisions.se	uptime.se	dumpkstats.se
	net_example	nproc.se	kvmname.se



System Accounting

- Accounting is not enabled by default.
- Accounting enables you to track what is executed and how much resource it uses.
- It can be very useful in determining what is using your system.
- You must run the reports to reduce the data.
- It adds system overhead.
- If you have a varying environment, it is the easiest way to decide where to spend your time.
- You can correlate accounting data with tuning reports.

System Accounting

The system performance monitors provide information on the state of the system resources, not on the activity of individual processes.

To get individual process information, such as memory and CPU usage, run times, and I/O counts, system accounting must be used. Accounting is not enabled by default, due to processing overhead, and must be enabled specifically when desired.

The accounting system collects large amounts of data, which then must be processed to extract and format the information of interest. Accounting data can be very useful in determining the most frequently used or the most expensive (in terms of resource usage) programs in your environment, to help focus your tuning efforts.

A detailed description of the use of the accounting system is given in Appendix C.



Sun Educational Services

Viewing Tuning Parameters

Several utilities that provide the current settings of system variables are:

- sysdef
- adb
- ndd

Viewing Tunable Parameters

There are several utilities that provide the ability to look at the current values of system tuning parameters.

The `sysdef` command, near the end of its output, provides a list of approximately 60 of the most commonly used tuning parameters.

The kernel debugger, `adb`, enables you to look at (and alter) the values of the tuning parameters. While not very user friendly, it and the similar `crash` utility allow the inspection of the running system.

The `ndd` utility enables you to look at and change the settings of network device driver and protocol stack parameters. Most of these parameters are undocumented, so caution should be used when changing them. Changes to these parameters may also be made in `/etc/system`.



Setting Tuning Parameters

- Parameters are usually set in /etc/system
 - The system must be rebooted for changes to take effect.
- Some changes can be made with adb while the system is running.
 - Be careful – you could take the system down.
 - Not all parameters can be changed with adb.
- Network device parameters can be set and inspected with ndd.
 - These changes can also be made in /etc/system.

Setting Tunable Parameters

Changes to tuning parameters are usually made in the /etc/system file. The /etc/system file is read just after the kernel has been loaded into memory. The parameters specified in it are set in the appropriate symbol location in the kernel (and other kernel resident modules), and system initialization continues. The system must be rebooted for changes to /etc/system to take effect.

Changes may also be made to a running system using adb. Not all parameters can be changed this way, however. If a parameter is used at boot time to specify the size of a table, for instance, changing it after the boot will have no effect. Without a knowledge of system internals, it can be difficult to determine whether the change will be seen.

And, as mentioned before, network device and protocol stack parameters can be changed for the life of the boot by using ndd, and "permanently" by using the /etc/system file.



/etc/system

- This file can be used to modify kernel-tunable variables.
- The basic format is:
`set [modulename :]variablename = value`
- /etc/system can also be used to force modules to be loaded at boot time, to specify a root device other than the default, and so on.
- The system must be rebooted for changes to take effect.
- Care should be used when updating /etc/system; this file modifies the operation of the kernel.

/etc/system

The /etc/system file can be used to set the values for kernel-resident tuning and option parameters.

Values may be specified for parameters in modules other than the kernel by specifying the proper module name in front of the parameter name. If no module name is given, a parameter in the kernel is assumed. It does not matter which of the kernel files (unix or genunix) that the parameter resides in.

If the /etc/system file is changed incorrectly, the system may be unable to boot. Non-existent tuning parameters will cause warning messages and syntax errors may prevent the system from booting. If this should occur, boot the system with the -as options, and specify a file such as /dev/null as the location for the /etc/system file. Once you reach single-user mode, correct the error(s) and reboot.

Check Your Progress

Before continuing on to the next module, check that you are able to accomplish or answer the following:

- Describe how performance data is collected in the system
- Describe how the tools retrieve the data
- List the data collection tools provided by the Solaris OS
- Explain how to view and set tuning parameters
- Describe the system accounting functions
- Describe the use of the SyMON system monitor

Think Beyond

Some questions to ponder are:

- What other tools might the Solaris environment need?
- Why do the tools seem to overlap so much?
- Can you write your own tools?
- Can an industry-standard set of tools be developed?

Objectives

Upon completion of this lab, you should be able to:

- Execute performance monitoring tools native to the Solaris environment
- Start system accounting
- Install the SE Toolkit
- Install `memtool`

Tasks

Using the Tools

Complete the following steps:

1. Change to the `lab2` directory for this lab.
2. Log in at the command line as `root`.
3. List which tool(s) you would use to look at the following data elements:

Data Element	Tool(s)	Option(s)
Tape device response time		
Free memory		
Per CPU interrupt rates		
Network collision rates		
UFS logical reads		
NFS read response time		
I/O device error rates		
Semaphore activity		
Active sockets		

4. Using the man pages for those tools, determine which option(s) you would need for each data type.

Do not be concerned with the meaning of these elements. They will be covered later in the course.

5. Open two terminal windows on your workstation. Start one of the tools you listed in step 1 in one window and another tool in the second window. Use a 15-second interval, and let the output display on the screen.
6. Open another terminal window and run the `load3.sh` script which is in your `lab2` directory. (This could take quite a while on a Solaris 7 platform).

When the script finishes, look at the script. Based on what it does, does the output from the tuning tools that you were running in step 3 make sense?

7. Stop the tuning tools that you started in step 3.

Enabling Accounting

Complete these steps:

1. Using the directions given in Module 2, enable accounting for your system (see Appendix C, page C-8).
2. As `root` user, enter:

```
crontab -e sys
```

This starts a `vi` session with the `sys` account's `crontab` file. Uncomment the commands in this `crontab` file and save it.

3. Reboot your system.

Installing the SE Toolkit

The SE Toolkit is provided in the `perf` directory of your student account. To install it on your lab system, as `root`:

1. Change to the `perf` directory.
2. Uncompress and untar the toolkit installation files using:

```
zcat RICHPse.tar.Z | tar -xvf -  
zcat RICHPsex.tar.Z | tar -xvf -  
zcat ANCrules.tar.Z | tar -xvf -
```

3. Add the three packages now found in the directory using the following command. Respond `y` to all prompts about commands to be started at boot time.

```
pkgadd -d . RICHPse RICHPsex ANCrules
```

Installing memtool

memtool is provided in the memtool directory of your student account. To install it on your lab system, as root:

1. Change to the memtool directory. Uncompress and untar the toolkit installation files using:

```
zcat RMCmem3.7.5Beta1.tar.Z | tar -xvf -
```

2. Add the memtool package using:

```
pkgadd -d .
```

3. Respond **y** to all prompts about commands to be started at boot time.

4. After successful installation, manually load the kernel module.

```
/opt/RMCmem/driv/bunyipload
```

5. Verify the kernel module was successfully loaded.

```
# modinfo | grep bunyip
```

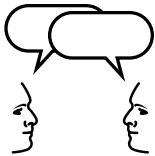
```
209 102bff10 3659 - 1 bunyipmod (MemTool Module %I% %E%)
```

Objectives

Upon completion of this module, you should be able to:

- Describe the lifetime of a process in the SunOS environment
- Explain multithreading and its benefits
- Describe locking and its performance implications
- List the functions of the clock thread
- Explain the process and thread tuning data
- Describe the related tuning parameters

Relevance



Discussion – The following questions are relevant to understanding the content of this module:

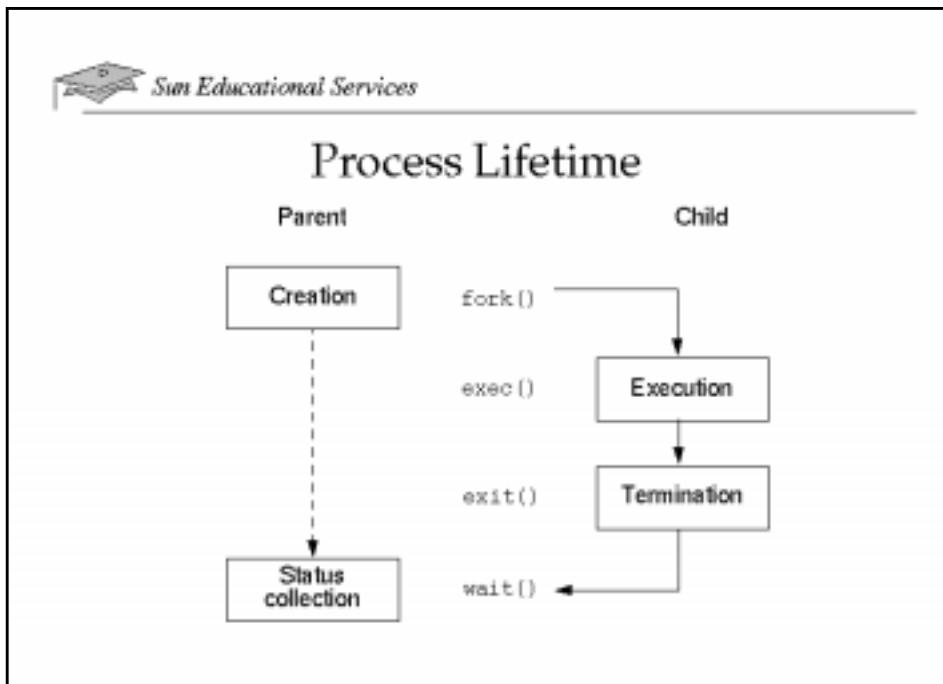
- What is a thread? Why are they used?
- Which is tunable for processes and threads?
- Can any tuning be done without programmer intervention?

Additional Resources



Additional resources – The following references can provide additional details on the topics discussed in this module:

- Graham, John. 1995. *Solaris 2.x Internals*. McGraw-Hill.
- Cockcroft, Adrian, and Richard Pettit. 1998. *Sun Performance and Tuning*. Palo Alto: Sun Microsystems Press/Prentice Hall.
- *Solaris 2.6 System Administration Guide, Tuning Kernel Parameters*.
- *Solaris 2.6 AnswerBook, Solaris NFS Administration Guide*.
- Man pages for the various commands (`lockstat`, `clock`, `ps`, `acctcomm`, `lastcomm`) and a performance monitoring tool (`sar`).



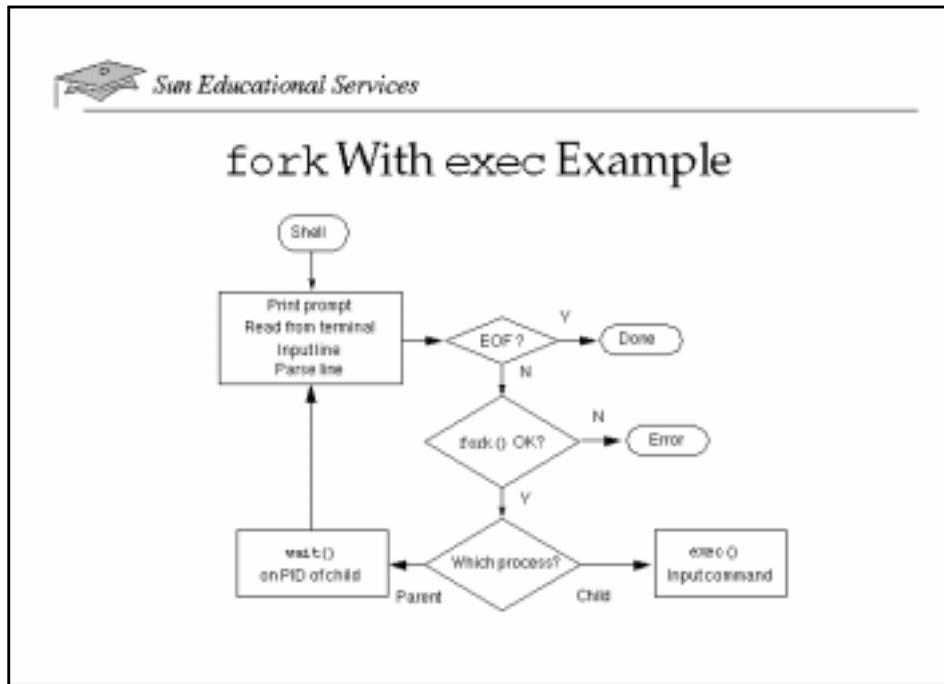
Process Lifetime

All processes in the system, except for `sched` (process 0), are created by the `fork` system call. The `fork` call makes an exact copy of the calling process, and the parent, in a new virtual address space (the child).

The child then usually issues an `exec` system call, which replaces the newly created process address space with a new executable file. The child continues to run until it completes its job. It then issues an `exit` system call, which returns the child's resources to the system.

While the child is running, the parent either waits for the child to complete, by issuing a `wait` system call, or continues to execute. If it continues it can check occasionally for the child's completion, or be notified by the system when the child exits.

There is no practical limit to the number of `fork` requests a process may issue (except for system resources), and the child process may also issue `fork` requests.



fork With exec Example

The example in the above overhead shows how a shell would process an input command. After processing the command, the shell issues a `fork` to create a copy of itself. The copy then issues an `exec` call to run the requested command.

The shell then either waits for the command to finish, or if background execution was requested (with an &), issues the next prompt and waits for another input command.



Process Performance Issues

- The maximum number of processes allowed is 30,000.
- Executables that use shared libraries take fewer system resources (disk space, memory, I/O, and so on).
- Threads are more efficient than multiple processes.
- Zombie processes cause no performance problems.

Process Performance Issues

The system for internal reasons currently allows a maximum of 30,000 simultaneous processes. (Setting the number of processes is discussed later in this module.)

You can also save memory by remembering that executables which use shared libraries take fewer system resources (memory, swap, and so on) to use. This is discussed in Module 5.

If your program needs multiple processes, consider using threads. As discussed later in this module, they can save resources and improve performance.

Do not be concerned about the resource use of zombie processes. Other than taking an available process table slot, they consume no other resources.



Process Lifetime Performance Data

The `sar -v` and `-c` options provide data related to the process lifetime.

- `scall/s` – Total number of system calls made by all processes in the system, both user and system
- `fork/s` – Total number of `fork` system calls
- `exec/s` – Total number of `exec` system calls
- `proc-sz` – Percentage of the total number of allowed processes that are in use

Process Lifetime Performance Data

The `sar` command is the only performance monitor that provides usable process lifetime performance data. The most useful of the available information is shown in the overhead image above. Sample output of the `sar -v` command is shown below. The `-v` option reports the status of processes, i-nodes, and file tables.

```
# sar -v
```

```
SunOS rafael 5.7 Generic sun4u      06/10/99
```

00:00:00	proc-sz	ov	inod-sz	ov	file-sz	ov	lock-sz
01:00:00	66/922	0	3303/4236	0	505/505	0	0/0
02:00:00	66/922	0	3303/4236	0	504/504	0	0/0
...							
...							
14:40:02	65/922	0	4813/4813	0	512/512	0	0/0
15:00:02	68/922	0	4572/4572	0	516/516	0	0/0

```
#
```

This output contains the following fields related to processes:

- **proc-sz** – The number of process entries (proc structs) currently being used and the number of processes allocated in the kernel (*max_nprocs*).

Sample output of the **sar -c** command is shown below. The **-c** option reports system call information.

```
# sar -c
```

```
SunOS rafael 5.7 Generic sun4u      06/10/99

00:00:00 scall/s sread/s swrit/s   fork/s   exec/s rchar/s wchar/s
01:00:00      406      5      3    0.01    0.01    1175     628
02:00:00      405      5      3    0.01    0.01    1170     626
03:00:00      413      6      4    0.05    0.05    1422     811
...
...
14:40:02     1018     69     45    0.07    0.08    35065   28775
15:00:02     1172    114     76    0.02    0.02    71670   51751

Average       621      30     20    0.02    0.02    20827   16167
#
```

This output contains the following fields related to system calls:

- **scall/s** – All types of system calls per second.
- **fork/s** – The number of fork system calls per second (about 0.5 per second on a four- to six-user system). This number will increase if shell scripts are running.
- **exec/s** – The number of exec system calls per second. If **exec/s** divided by **fork/s** is greater than three, look for inefficient *PATH* variables.

 Sun Educational Services

Process-Related Tunable Parameters

- All are scaled from *maxusers*

Name	Default	Min	Max
<i>maxusers</i>	Number of Mbytes of real memory / 2	8	1024 (2048)
<i>nproc_nprocs</i>	(16 <i>maxusers</i>) + 10	138	30000
<i>nproc_nprocs</i>	<i>max_nprocs</i> + 5	133	29995
<i>nquot_nquot</i>	(<i>maxusers</i> x 10) + <i>max_nprocs</i>	213	50480
<i>pty_npty</i>	48	48	3000+
<i>pt_nt</i>	48	48	3000+
<i>ncsize_ncsize</i>	4 x (<i>maxusers</i> + <i>max_nprocs</i>) + 320	236	34906
<i>afs_ninode</i>	4 x (<i>maxusers</i> + <i>max_nprocs</i>) + 320	236	34906

Process-Related Tunable Parameters

The majority of tuning parameters related to the process lifetime are scaled from the *maxusers* parameter by default.

The problem of scaling the size of several kernel tables and buffers is addressed by creating a single sizing variable. The variable name, *maxusers*, was related to the number of time-sharing terminal users the system could support. Today, there is no direct relationship between the number of users a system supports and the value of *maxusers*. Increases in memory size and complexity of applications require much larger kernel tables and buffers for the same number of users. The *maxusers* setting in the Solaris 2.x OS is automatically set via the *physmem* variable. The minimum limit is 8 and maximum is 1024, corresponding to systems with 1 Gbyte or more of random access memory (RAM). *maxusers* can be set manually in */etc/system*, but manual setting is checked and limited to a maximum of 2048.

max_nprocs is scaled from *maxusers*, and specifies the number of processes that the system may have simultaneously active. While setting *max_nprocs* too high does not waste memory, lowering it may provide some protection against system over-utilization.

maxuprc is the maximum number of processes a single user may create. The root user is not limited by this. The variable is usually set far too large but unless you have a runaway program, there should be no need to change it.

ndquot specifies the number of disk quota structures to build. You can ignore this value if you are not using disk quotas; if you are using disk quotas, check the *quot* man page to determine its proper value.

npty and *pt_cnt* specify the number of SunOS 4 Berkeley Software Distribution (BSD) and SunOS 5 (System V) pseudo-terminals, respectively, to create on the system. Over 3000 have been tested, and more than 8000 have been created; there is no practical limit to these devices. If you increase the number of either, you must use the *boot -r* command before the new pseudo-terminals are usable.

ncsize and *ufs_ninode* are used to specify the number of entries in the directory name lookup cache (DNLC) and UFS *inode* cache, respectively. These are discussed in Module 9.



Multithreading

- A thread is a logical sequence of program instructions.
- The kernel is *multithreaded*.
 - Multiple tasks may be running in the kernel simultaneously and independently.
- A user process can have many application threads that execute independently of each other.
 - There is still only one executable in the process.
- Fewer system resources are used than multiple processes.
- Special programming techniques are required.

Multithreading

A *thread* is a logical sequence of instructions in a program that accomplishes a particular task. In a multithreaded program, multiple threads are created at the programmer's request, each performing its assigned task. This technique allows concurrent execution in the same process, making much more efficient use of system resources.

Testing has shown that the creation of a process takes 33 times more CPU time than creating a new thread.

The kernel is also multithreaded, allowing multiple tasks to execute at once. This dramatically improves performance, as systems with multiple CPUs do not have to execute kernel code sequentially.

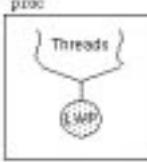
When using threads, a user process still has only one executable. Each thread has its own stack, but they share the same executable file.

Since application threads share the process address space and data, care must be taken to *synchronize* access to shared data. The threads are invisible to the operating system, so there is no system protection between application threads. Some amount of lightweight process (LWP) information is available using the `ps -L` command.

While there are significant resource saving and performance benefits to using threads, it does require special programming techniques that many programmers are not comfortable with. However, similar techniques are required to synchronize multiple processes, so it is not difficult to learn.

 Sun Educational Services

Application Threads



- Each thread has its own stack.
- Threads share the process address space.
- The threads execute independently (and concurrently).
- Threads are *completely* invisible outside of the process.
- Threads cannot be controlled from the command line.

Application Threads

Each thread has its own stack and some thread local storage. All of the address space is visible to all of the threads. Unless programmed that way, threads are not aware that there are other threads in the process.

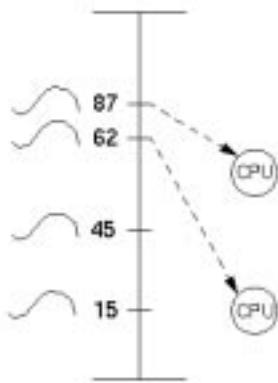
Threads are invisible outside of the process. There are no commands that will create, kill, change the priority of, or even show the existence of application threads. They are *completely* invisible outside of the process.

Threads are scheduled independently of each other, and can execute concurrently. The programmer controls the number of threads that can execute concurrently.

Much of the state of the process is shared by all of the threads, such as open files, the current directory, and signal handlers.



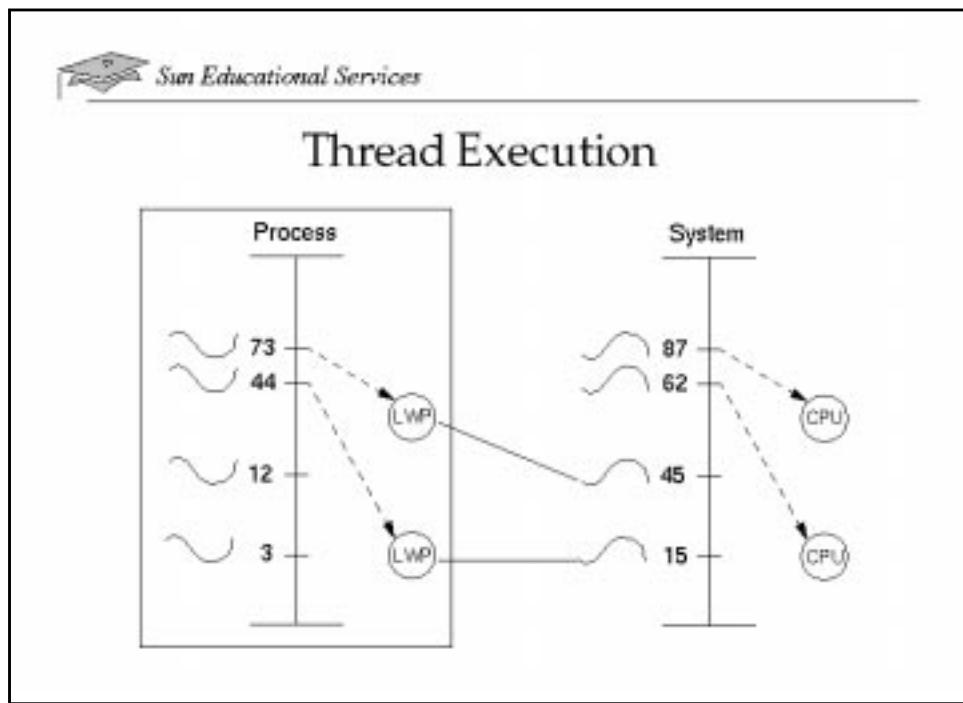
Process Execution



Process Execution

The system provides dispatch queues (discussed in Module 4) that queue the work to be executed by the CPUs in priority order.

As CPUs become free, kernel threads are taken from the dispatch queue and run.

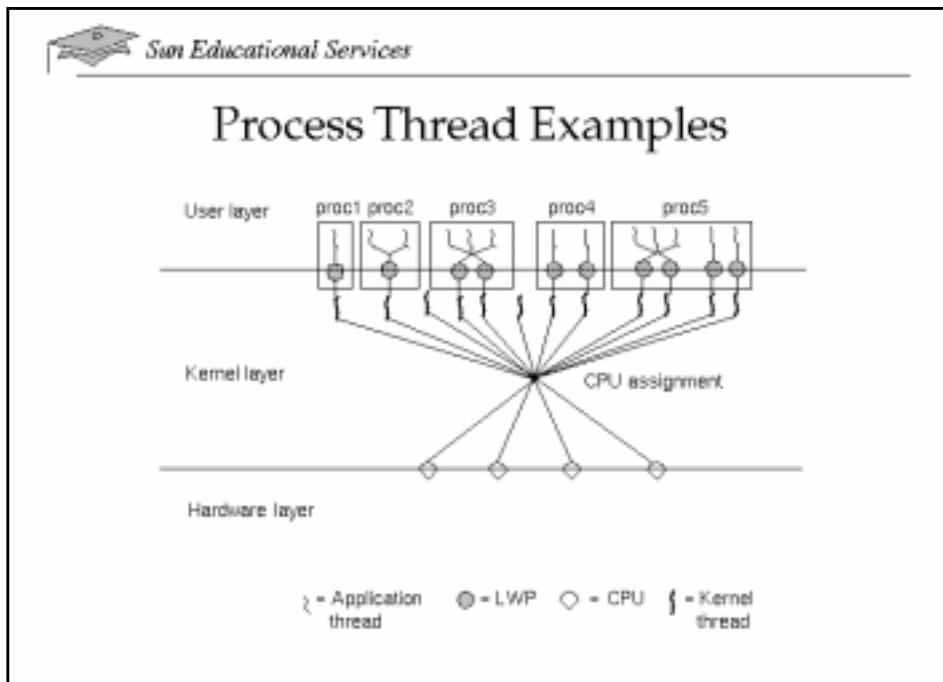


Thread Execution

Application thread execution operates similarly to that of the system CPUs. Application threads are queued on a single dispatch queue in the application process. They are not queued directly to the system CPU dispatch queues.

The system provides "logical" CPUs, called lightweight processes (LWPs), that perform the function of the system CPUs for the application threads. The application threads attach to an available LWP, and are then queued through a kernel thread connected to the LWP in the system's CPU dispatch queue.

This model has two symmetrical levels: a dispatch queue and "CPUs" for the application threads, and dispatch queues and real CPUs for the kernel threads. Since the number of LWPs is under programmer control, programmers can control the concurrency level of the threads in their application.



Process Thread Examples

The diagram in the above overhead shows examples (proc2 through proc5) of multithreaded applications. Note the top layer is in the user process; only the application sees the application threads.

The middle layer is in the kernel, which includes the LWP and the kernel thread structures. The kernel thread is the entity which is scheduled by the kernel.

The bottom layer represents the hardware; that is, the physical CPUs.

The kernel threads that are not attached to an LWP represent system daemons such as the page daemon, the callout thread, and the clock thread. These kernel threads are known as *system threads*.



Performance Issues

Multithreading an application allows it to:

- Be broken into separate tasks that can be scheduled and executed independently
- Take advantage of multiprocessors with less overhead than multiple processes
- Share memory without going through the overhead and complexity of IPC mechanisms
- Use a cleaner programming model for certain types of applications
- Extend the program more easily

Performance Issues

Multithreading an application allows it to execute much more efficiently, and provides many performance benefits. The application:

- Is broken into separate tasks that can be scheduled and executed independently
- Takes advantage of multiprocessors with less overhead than multiple processes
- Shares memory without going through the overhead and complexity of the interprocess communication (IPC) mechanisms
- Takes advantage of asynchronous I/O with low overhead locks
- Uses a cleaner programming model for certain types of applications



Thread and Process Performance

Creation primitives

	Microseconds	Ratio
Unbound threadcreate	62	-
Bound threadcreate	360	6.7
fork()	1700	32.7

- Using fork can create memory management issues.
- Timings were obtained on a SPARCstation™ 2 system.

Thread and Process Performance

The performance numbers in the above overhead were obtained on a SPARCstation™ 2 (Sun 4/75). The measurements were made using the built-in microsecond resolution, real-time timer.

The table shows the creation times for unbound threads, bound threads, and processes. It measures the time consumed to create a thread using the default stack that is cached by the threads package. It does not include the initial context switch to the thread.

The ratio column gives the ratio of the second and third rows with respect to the first row (the creation time of unbound threads).



Locking

- Locks are used to synchronize threads by serialization.
- They protect critical data from simultaneous *write* access.
- Locks must be used when threads share writable data.
- SunOS provides four types of locks.
 - Which type is used depends on the requirements.
- A bad locking design can cause performance problems.
- Locking problems usually require a significant reprogramming.

Locking

In a multithreaded environment, it is quite likely that two different threads will try to update the same data structure at the same time. This can cause serious problems, since the update made by one thread could be overwritten by another.

Locks are used to protect critical data structures from *simultaneous* update access. Locks can also provide a synchronization mechanism for threads.

Locks must be used only when threads share writable data. Often there is no necessity to lock the data just to look at it, which can improve performance, although frequent locking is required to write the data. There are several different locking mechanisms available to handle the different locking situations.

The Solaris environment provides four different types of locks:

- ▼ Mutexes
- ▼ Condition variables
- ▼ Counting semaphores (basic + System V)
- ▼ Multiple-reader, single-writer locks

A bad locking design, for example, protecting too much data with one lock, can cause performance problems by serializing the threads for too long.

Repairing these problems usually requires a significant redesign by the programmer. Identifying them can be difficult.



Locking Problems

- Lock contention
 - Granularity
 - Inappropriate lock type
- Deadlock
 - "Lost" locks
 - Race conditions
- Incomplete implementation

Locking Problems

Locking problems are difficult to identify, and usually appear as programs running much slower than expected, for no obvious reason.

Locking problems are almost always caused by programming errors, such as those listed in the overhead image above. They must be fixed by the programmer. In some rare cases, they can be caused by extremely heavy system loads, which again, must be fixed by the programmer.

Some locking problems, such as forgetting to free a lock, can be detected by the `locklint` utility, which is part of the Sun Workshop development tool set. This utility is only useful to the programmer.

However, any user can use the `lockstat` command to identify the source of a run-time locking problem.



The lockstat Command

- Lock use in the kernel is identified.
- Unidentified delays may be caused by lock contention.
- Excessive counts may indicate a problem.

```
# lockstat ipstat

Adaptive mutex block: 2 events

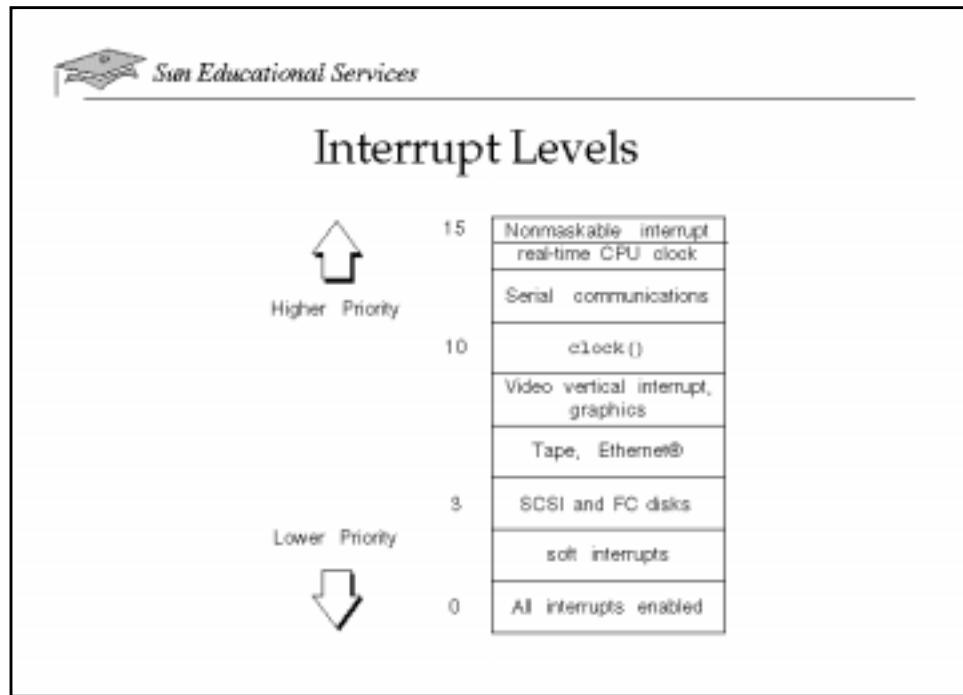
Count indv cuml rmt      nsec Lock          Caller
-----
=   1 50% 50% 1.00    87500 0xf5ae43d0
=   1 50% 100% 1.00   151000 0xf5ae3nb8
=   esp_poll_loop+0x8c
=   esp_poll_loop+0x8c
```

The lockstat Command

The **lockstat** command can be used to identify locking problems in a specific executable.

When run like **truss**, the **lockstat** command runs the specified command and monitors it for lock usage. It monitors locks used by the program within its own process (for example, with multithreading), as well as locks obtained for the process in the kernel. These locks are acquired during processing of a system call, for instance.

Large counts (**indv**), with a slow locking time (**nsec**) or very large locking times, can identify the source of the problem. While locking problems in the kernel must be fixed by the vendor, the lock name is usually prefixed with the module name. With the module name, known locking problems in the module can be searched for, and perhaps a patch identified.



Interrupt Levels

Just as there are priorities for threads in the system to access the CPU and for I/O devices, interrupts coming into the system also have priorities.

These priorities, usually assigned by the device driver, govern the order in which device interrupts are processed by the system. Interrupts are generally handled by a CPU on the system board that the I/O interface board is attached to.

Do not try to change the interrupt priority for a particular device. At certain levels there are programming restrictions on device drivers that lower priority drivers may not follow. Also, this can cause devices to get slower service than they require, causing overruns and retries, which slows system throughput.



The `clock` Routine

- The `clock` executes at interrupt level 10.
- Most system timing is run off this clock.
- Each time the `clock` routine executes is a *tick*.
- For most processors, there are 100 ticks per second.
 - Ticks per second can be set to 1000 for real-time processing
 - This is the limit of normal timing resolution.
- The time-of-day clock will run slow.

The `clock` Routine

The `clock` executes at interrupt level 10, a very high priority. There is one system thread per system that manages the `clock`, which is associated with a particular CPU.

Each time the `clock` routine executes is a tick. For SPARC processors (except sun4c systems), there are 100 ticks per second as shown by the `vmstat -i` command. This can be changed to 1000 ticks by requesting the high resolution rate. High resolution rate is used primarily for real-time processing. This is discussed in more detail in Module 4.

```
# vmstat -i
interrupt      total      rate
-----
clock        85912634      100
zsc0          3197273       3
zsc1          6013844       6
hmec0         5952964       6
fdc0            10          0
-----
Total        101076725     117
#
```



Clock Tick Processing

Every clock tick the system checks for:

- Timer time-outs
 - Sleep and wait times
 - I/O timeouts
 - Network timeouts
- Profiling timers
- Active threads which have used up their time slice
- Processes which have exceeded their CPU resource limit

Clock Tick Processing

For every clock tick, the system performs numerous checks for timer related work, some of which is shown in the overhead image above.

The system clock tends to drift slowly during operation. There are not always exactly 100 clock interrupts every second, due to interference from other system operations. Different systems drift at different rates.

To synchronize the system clocks with each other, and perhaps with an external exact time source, you can use the NTP (Network Time Protocol), provided with the Solaris 2.6 and above OS. See the man page for `xntpd(1M)` for more information.



Process Monitoring Using ps

- You need to identify active processes before determining:
 - Which process is causing a delay
 - Which resource is bottlenecking the process
- The `ps` command enables you to check the status of processes.
- The `ps` command helps determine how to set process priorities.
- The BSD version, `/usr/ucb/ps -aux`, provides the best performance related data.

Process Monitoring Using ps

Several methods exist for monitoring the process running on your system. This section gives examples of some of the different methods and also describes output fields when necessary.

Once you identify which processes are active, you can narrow it down to the one(s) causing the delay and determine what kind of resource is slowing down the process.

The `ps` command gives a high-level summary of the status of processes on your system. It has many options, but it is a relatively crude way to figure out what your processes are doing. The `ps` command enables you to check the status of active processes on a system, as well as display technical information about the processes. This data is useful for determining how to set process priorities. Setting process priorities are discussed further in Module 4.

Of all the available options, the best performance related summary comes from the BSD version of `/usr/ucb/ps -aux`, which collects all of the process data at one time, sorts the output by CPU usage, then displays the result. The unsorted versions of `ps` loop through the processes, printing as they go. This spreads out the time at which the processes are measured, and the last line process measured is significantly later than the first. By collecting at one time, the sorted versions give a more consistent view of the system. A quick look at the most active processes can be obtained easily using this command.

```
# /usr/ucb/ps -aux | head
USER      PID %CPU %MEM   SZ  RSS TT      S      START  TIME COMMAND
craigm    260  0.7 56.015049632672 ?
craigm    4234  0.4 15.021168 8744 pts/3
craigm    2549  0.4  8.735936 5048 ?
root      4901  0.2  1.9 1464 1096 pts/5
root      3     0.1  0.0    0   0 ?
craigm    4370  0.1  6.4 7280 3728 ???
root      183   0.0  2.7 2384 1544 ?
root      258   0.0  2.7 2288 1560 ?
root      0     0.0  0.0    0   0 ?
#
#
```

This summary immediately tells you who is running the most active processes.

- USER – User who started the process.
- PID – Process ID number assigned by the system.
- %CPU – Percentage of the CPU time used by the process.
- %MEM – Percentage of the total RAM in your system in use by the process (it will not add up to 100 percent as some RAM is shared by several processes).
- SZ – Amount of nonshared virtual memory, in Kbytes, allocated to the process. It does not include the program's text size which is normally shared.
- RSS – Resident set size of the process. It is the basis for %MEM and is the amount of RAM in use by the process.
- TT – Which "teletype" the user is logged in on.

- S – The status of the process.
 - ▼ S – Sleeping
 - ▼ O – Using cpu (on CPU) or running
 - ▼ R – Running and waiting for a CPU to become free
 - ▼ Z – Terminating but has not died (zombie)
 - ▼ P – Waiting for page-in
 - ▼ D – Waiting for disk I/O

Check disk performance and memory usage if many P or D statuses appear. This could indicate an overload of both subsystems.

- START – Time the process started up
- TIME – Total amount of CPU time the process has used so far
- COMMAND – Command being executed

The same basic information is displayed by the well-known freeware utilities `top` and `procstat`. The Solstice SyMON system monitor and most commercial performance tools also display this data.



Process Monitoring Using System Accounting

- Short-lived processes cannot be seen using `ps`.
- System accounting keeps a record of all processes.
- Terminating processes are logged to `/var/adm/pacct`.
- `acctcom` and `lastcomm` commands report on `/var/adm/pacct`.
- Summary scripts `/usr/lib/acct/runacct` and `monacct` create daily and monthly summary reports.

Process Monitoring Using System Accounting

Many processes live very short lives. You cannot see them with `ps`, but they may be so frequent that they dominate the load on your system. The only way to catch them is to ask the system to keep a record of every process that it runs, who ran it, what was it, when it started and ended, and how much resource it used. This is done by the system accounting subsystem.

Accounting data is most useful when measured over a long period of time. This can be useful on a network of workstations as well as on a single time-shared server. From this you can identify how often programs run, how much CPU time, I/O, and memory each program uses, and what work patterns throughout the week look like.

Refer to Appendix C, page C-8, for instructions on starting and stopping system accounting.

Once system accounting is enabled, as each program terminates, the kernel (the `exit` function) places an entry in the `/var/adm/pacct` file. The contents of this file can then be displayed using the `acctcom` command or the `lastcomm` command.

The **acctcom** command without options searches the **/var/adm/pacct** file and reports on processes that have terminated.

```
# acctcom
COMMAND
NAME    USER    TTYNAME      START TIME      END TIME      REAL (SECS) CPU (SECS) MEAN SIZE(K)
#accton root    ?          02:30:01 02:30:01   0.03  0.01  656.00
turnacct adm    ?          02:30:01 02:30:01   0.21  0.01  608.00
mv      adm    ?          02:30:01 02:30:01   0.08  0.01  656.00
closewtmp adm   ?          02:30:01 02:30:01   0.06  0.01  664.00
cp      adm    ?          02:30:01 02:30:01   0.05  0.01  592.00
...
...
date    sys    ?          13:40:00 13:40:00   0.01  0.01  520.00
sadc    sys    ?          13:40:00 13:40:00   0.17  0.03  709.33
#sh     sys    ?          13:40:00 13:40:00   0.41  0.07  1034.29
#sendmail root   ?          13:43:26 13:43:26   0.21  0.02  1372.00
vi      craigm ?          12:02:35 13:52:34  6599.68  0.90  1341.87
#
#
```

Refer to Appendix C, pages C-21 to C-23, for additional information on the **acctcom** command.

The **lastcomm** command without options also searches the **/var/adm/pacct** file and prints the last commands executed, in reverse order.

```
# lastcomm
acctcom    craigm ??        0.98 secs Thu Jun 10 13:52
vi         craigm ??        0.90 secs Thu Jun 10 12:02
sendmail SF root   —       0.02 secs Thu Jun 10 13:43
sh         S  sys   —       0.07 secs Thu Jun 10 13:40
sadc       sys    —       0.03 secs Thu Jun 10 13:40
date       sys    —       0.01 secs Thu Jun 10 13:40
ps         craigm ??        0.07 secs Thu Jun 10 13:38
...
...
closewtmp adm    —       0.01 secs Thu Jun 10 02:30
mv         adm    —       0.01 secs Thu Jun 10 02:30
turnacct   adm   —       0.01 secs Thu Jun 10 02:30
accton    S  root   —       0.01 secs Thu Jun 10 02:30
#
#
```

Refer to Appendix C, page C-29, for more information on the **/var/adm/pacct** file.

Some crontab entries must also be added to summarize and checkpoint the accounting logs. Collecting and checkpointing the accounting data itself puts a negligible additional load onto the system, but the summary scripts that run once a day or once a week can have a noticeable effect, so they should be scheduled to run out of hours.

The summary script `/usr/lib/acct/runacct` summarizes the raw data collected over the day, deletes the raw data file, and creates the daily summary files and daily reports in the `/var/adm/acct/sum` directory. Refer to Appendix C, pages C-20-23 for more information on the `runacct` script.

The summary script `/usr/lib/acct/monacct` creates an overall total from the day's totals and creates a summary report for the entire accounting period. The monthly analysis and summary files are stored in the `/var/adm/acct/fiscal` directory. Refer to Appendix C, page C-23, for more information on the `monacct` script.

Following is the first few lines from the `/var/adm/acct/fiscal/fiscrpt06` file:

```
Jun 1 07:30 1999 TOTAL COMMAND SUMMARY FOR FISCAL 06 Page 1
```

COMMAND NAME	NUMBER CMDs	TOTAL KCOREMIN	TOTAL COMMAND SUMMARY							CHARS TRNSFD	BLOCKS READ	
			TOTAL CPU-MIN	TOTAL REAL-MIN	MEAN SIZE-K	MEAN CPU-MIN	HOG FACTOR					
TOTALS	17014	1610658.50	100.21	704471.91	16072.67	0.01	0.00	1773230720		24821		
Xsun	1	1178599.41	33.38	11894.78	35307.14	33.38	0.00	-2092957760		486		
netscape	147	168303.52	11.99	29094.64	14037.96	0.08	0.00	1595135616		5950		
dtaction	941	138657.49	29.66		32.81	4674.14	0.03	0.90	999665872		59	
maker5X.	8	49693.35	5.11	8245.46	9717.75	0.64	0.00	280928256		25		
java	2	24237.40	1.70	31.59	14262.89	0.85	0.05	12537856		7		
acroread	27	11104.31	1.28	10090.15	8705.85	0.05	0.00	396662496		56		
dtwm	1	7553.02	1.49	11894.78	5070.28	1.49	0.00	79822850		133		
nsgmls	59	5035.41	1.22		2.08	4111.66	0.02	0.59	19720064		69	

Some of the fields in this output are:

- KCOREMIN – The product of the amount of CPU time used and the amount of RAM used while the command was active. The output is sorted by KCOREMIN.
- CPU-MIN – The number of minutes of CPU time.
- REAL_MIN – The elapsed time for the commands.

- SIZE-K – An average value for the resident set size (RSS) over the active lifetime of the process. It does not include times when the process was not actually running.
- HOG FACTOR – The ratio of CPU-MIN to REAL-MIN. A high factor means that this command hogs the CPU whenever it is running.
- CHARS TRNSFD – The number of characters read and written.
- BLOCKS READ – Data read from block devices (basically local disk file system reads and writes).

Process Monitoring Using SE Toolkit Programs

msacct.se

The **msacct.se** program uses microstate accounting to show the detailed sleep time and causes for a process. This program must be executed by root.

```
# /opt/RICHPSse/bin/se /opt/RICHPSse/examples/msacct.se

Elapsed time      67:37:19.092 Current time Thu Jun 10 11:19:15 1999
User CPU time      0.000 System call time    67:37:19.092
System trap time    0.000 Text pfault sleep     0.000
Data pfault sleep    0.000 Kernel pfault sleep     0.000
User lock sleep      0.000 Other sleep time     0.000
Wait for CPU time    0.000 Stopped time     0.000

Elapsed time      67:37:29.088 Current time Thu Jun 10 11:19:25 1999
User CPU time      0.000 System call time    67:37:29.088
System trap time    0.000 Text pfault sleep     0.000
Data pfault sleep    0.000 Kernel pfault sleep     0.000
User lock sleep      0.000 Other sleep time     0.000
Wait for CPU time    0.000 Stopped time     0.000
^C#
```

pea.se

The **pea.se** program also uses microstate accounting. This program runs continuously and reports on the average data for each active process in the measured interval (10 seconds by default). The data includes all processes and shows their average data since the process was created.

```
# /opt/RICHPSse/bin/se /opt/RICHPSse/examples/pea.se
tmp = pp.lasttime<9.29046e+08>
tmp_tm = localtime(&tmp<929045620>)
debug_off()

14:13:40 name lwmx pid ppid uid usr% sys% wait% chld% size rss pf
init      1   1   0   0  0.00  0.00  0.00  0.00  736 152 0.0
utmpd     1  219   1   0  0.00  0.00  0.00  0.00  992 536 0.0
ttymon    1  259  252   0  0.00  0.00  0.00  0.00 1696 960 0.0
sac       1  252   1   0  0.00  0.00  0.00  0.00 1624 936 0.0
rpcbind   1  108   1   0  0.00  0.00  0.00  0.00 2208 840 0.0
automountd 13  158   1   0  0.00  0.00  0.00  0.00 2800 1808 0.0
```

Some of the fields of this output are:

- name – The process name
- lwmx – The number of LWPs for the process so you can see which are multithreaded
- pid, ppid, uid – Process ID, parent process ID, and user ID
- usr%, sys% – User and system CPU percentage measured accurately by microstate accounting
- wait% – Process time percentage spent waiting in the run queue or for page faults to complete
- chld% – CPU percentage accumulated from child processes that have exited
- size – Size of the process address space
- rss – Resident set size of the process, the amount of RAM in use by the process
- pf – Page fault per second rate for the process over the interval

The pea.se program also has a wide mode which displays additional information. Use the -DWIDE option to the se command

```
/opt/RICHpse/bin/se -DWIDE /opt/RICHpse/examples/pea.se.
```

The additional data includes:

- Input and output blocks per second
- Characters transferred by read and write calls
- System call per second rate over the interval
- Voluntary context switches where the process slept for a reason
- Involuntary context switches where the process was interrupted by higher priority work or exceeded its time slice

ps-ax.se

The ps-ax.se program is a /usr/ucb/ps -ax clone. The program monitors only processes that you own. This program must be executed by root.

```
# /opt/RICHpse/bin/se /opt/RICHpse/examples/ps-ax.se
  PID TT      S TIME COMMAND
    0 ?      T 0:00 sched
    1 ?      S 0:01 /etc/init -
    2 ?      S 0:00 pageout
    3 ?      S 1:16 fsflush
   219 ?      S 0:00 /usr/lib/utmpd
...
...
1282 pts/2      S 0:00 dtpad -server
1433 pts/4      S 0:00 -sh
4370 ?          S 0:01 /usr/dt/bin/dtterm
2529 ?          S 0:00 /usr/dist/pkgs/cam,v1.5/5bin.sun4/cam netscape http://h
#
#
```

ps-p.se

The ps-p.se program is a ps -p clone (using /usr/ucb/ps output format). The program monitors only processes that you own.

```
# /opt/RICHpse/bin/se /opt/RICHpse/examples/ps-p.se 4234
  PID TT      S TIME      COMMAND
  4234 pts/3      S 4:12.73 /usr/dist/share/framemaker,v5.5.6/bin/sunxm.s5.sparc/ma
#
#
```

pwatch.se

The pwatch.se program watches a process as it consumes CPU time. It defaults to process ID 3 to watch the system process fsflush. It monitors only processes that you own.

```
# /opt/RICHpse/bin/se /opt/RICHpse/examples/pwatch.se
fsflush has used 77.3 seconds of CPU time since boot
fsflush used 0.0 %CPU, 0.0 %CPU average so far
fsflush used 0.2 %CPU, 0.1 %CPU average so far
^C#
```

Check Your Progress

Before continuing on to the next module, check that you are able to accomplish or answer the following:

- Describe the lifetime of a process in SunOS
- Explain multithreading and its benefits
- Describe locking and its performance implications
- List the functions of the clock thread
- Explain the process and thread tuning data
- Describe the related tuning parameters

Think Beyond

Some questions to ponder are:

- Why does the Solaris environment need threads when processes have worked for so long?
- How many threads can a process have? Why is that the limit?
- Why are LWPs used in the thread model? Are they really needed?
- Why are locks difficult for programmers to work with?

Objectives

Upon completion of this lab, you should be able to:

- Understand the relationship of *maxusers* to other tuning parameters
- Understand output from the `lockstat` command
- Understand various methods of displaying process information

Tasks

Using maxusers

The `maxusers` parameter is used to set the size of several system tables. Some of the tables that are modified directly by `maxusers` are:

- `max_nprocs` – Maximum number of processes
- `ufs_ninode` – Size of the inode cache
- `maxuprc` – Maximum number of processes per user
- `ncsize` – Size of the directory name lookup cache

Complete the following steps:

1. Change to the `lab3` directory for this lab.
2. Log in at the command line as `root`.
3. Type `./maxusers.sh`. This lists the current value of `maxusers` and some of the parameters it affects. Record the values.

`maxusers` _____

`max_nprocs` _____

`ufs_ninode` _____

`maxuprc` _____

`ncsize` _____

4. Run the `sysdef` command and save the output to a file.
5. In the `/etc/system` file, add 50 to the current value of `maxusers`.

`set maxusers=new_value`

6. Reboot the system.
7. Verify the changes by using the `./maxusers.sh` script.

8. Run the `sysdef` command again, saving the output to a file different from that used in step 6. Run `diff` on the two files to identify any changes in the `sysdef` output.
9. Restore the `/etc/system` file to its original form and reboot the system.

Using lockstat

Complete the following step:

1. To see output from `lockstat`, start the HotJava™ browser under `lockstat`. Use the command:

```
% lockstat hotjava
```

Remember, you will not see any output from `lockstat` until HotJava exits.

Monitoring Processes

Complete the following steps:

1. Display the most active processes on your system. Use the command:

```
# /usr/ucb/ps -aux | head
```

2. Answer the following questions:

▼ What is the most active process on your system?

▼ What percentage of CPU time is used by the most active process on your system?

▼ What percentage of the total RAM in your system is in use by the most active process?

- ▼ Are there any processes that are blocked waiting for disk I/O?
How can you tell?

- ▼ Is your system doing any unnecessary work? Can any unused daemons be disabled? How would you prevent unused daemons from starting at boot time?

- ▼ What is the advantage of using system accounting to monitor processes over the `ps` command?

3. Use the `acctcom` command to display processes that have terminated.
4. Use the `lastcomm` command to display the last commands executed, in reverse order.

5. Answer the following questions:

- ▼ What is the purpose of the system accounting /usr/lib/acct/runacct summary script?
-

- ▼ What is the purpose of the system accounting /usr/lib/acct/monacct summary script?
-

6. Monitor processes using the following SE Toolkit programs:

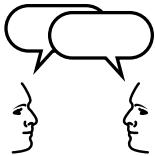
- ▼ msacct.se
- ▼ pea.se
- ▼ ps-ax.se
- ▼ ps-p.se
- ▼ pwatch.se

Objectives

Upon completion of this module, you should be able to:

- Describe the Solaris scheduling classes
- Explain the use of dispatch priorities
- Describe the time-sharing scheduling class
- Explain Solaris real-time processing
- List the commands that manage scheduling
- Describe the purpose and use of processor sets
- Introduce Solaris Resource Manager™

Relevance



Discussion – The following questions are relevant to understanding the content of this module:

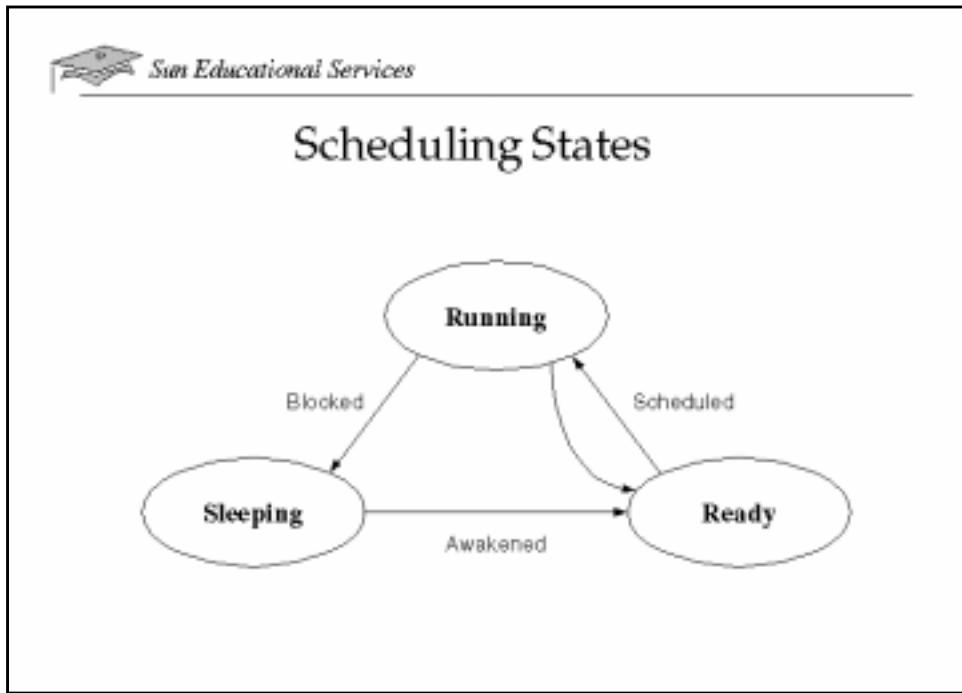
- How do you control how the system prioritizes its work?
- Can CPUs be dedicated to certain work?
- Do the tuning tasks differ with large numbers of CPUs?

Additional Resources



Additional resources – The following references can provide additional details on the topics discussed in this module:

- Cockcroft, Adrian, and Richard Pettit. 1998. *Sun Performance and Tuning*. Palo Alto: Sun Microsystems Press/Prentice Hall.
- Graham, John. 1995. *Solaris 2.x Internals*. McGraw-Hill.
- Man pages for the various commands (`priocntl`, `dispadmin`, `psrset`, `psrinfo`, `psradm`, `psrset`, `prtdiag`, `prtconf`, `sysdef`) and performance tools (`sar`, `vmstat`, `mpstat`).
- "Solaris Resource Manager, Controlling System Resources Effectively." Whitepaper, Sun Microsystems, Inc., 1995.



Scheduling States

A kernel thread can exist in one of three states:

- Running – Actually executing on a CPU
- Ready – Waiting for access to a CPU
- Sleeping – Waiting for an event, such as an I/O interrupt, wait time period expiration (such as a `sleep`), or lock availability

A thread moves between these states depending on the work being performed by the program that it is supporting.

Remember that application threads do not queue for the CPU directly, although system threads do. They are attached to an LWP and kernel thread pair. The kernel thread is then assigned to a dispatching queue when there is work to do for the application thread.



Scheduling Classes

- Four scheduling classes are provided by SunOS:
 - Timesharing – TS
 - Interactive – IA
 - System – SYS
 - Real-time – RT
- The classes are loaded dynamically as they are used.
- Scheduling algorithms are table driven.

Scheduling Classes

SunOS provides four scheduling classes by default. These are:

- TS – The timesharing class, for normal user work. Priorities are adjusted based on CPU usage.
- IA – The interactive class, derived from the timesharing class, provides better performance for the task in the active window in OpenWindows™ or CDE.
- SYS – The system class, also called the kernel priorities, is used for system threads such as the page daemon and clock thread.
- RT – The real-time class has the highest priority in the system, except for interrupt handling; it is even higher than the system class.

Like device drivers and other kernel modules, the scheduling class modules are loaded when they are needed.

Scheduling class behavior is controlled by root-adjustable tables.

Scheduling Classes

Class Characteristics

Timesharing/Interactive

The TS and IA classes have the following characteristics:

- Time sliced – The CPU is shared in rotation between all threads at the same priority.
- Mean time to wait scheduling – Priorities of CPU-bound threads are lowered; priorities of I/O-bound threads rise.

System Class

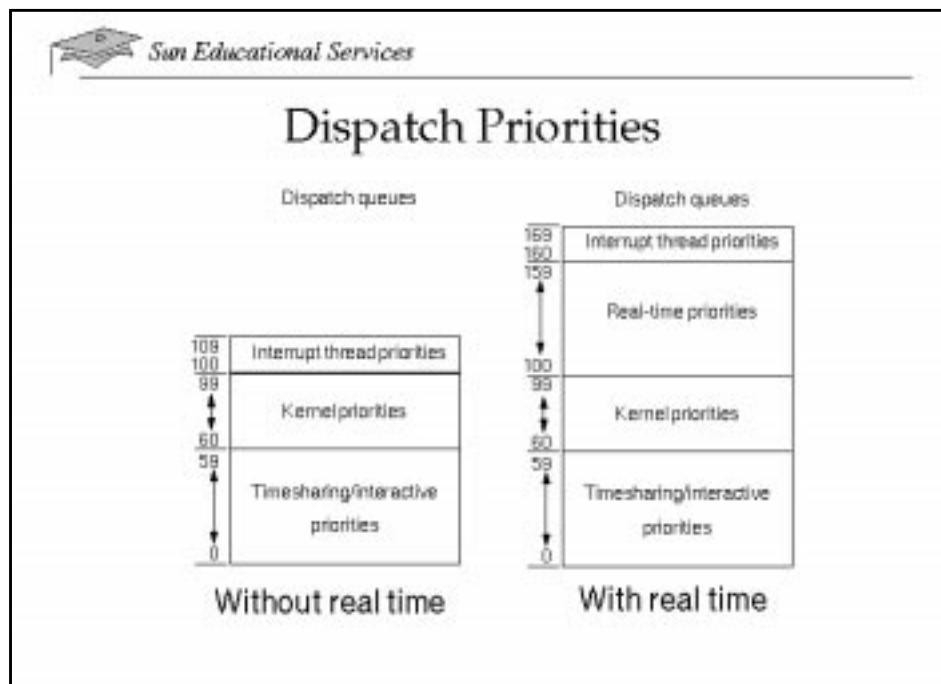
The SYS class has the following characteristics:

- Not time sliced – Threads run until they finish their work or are preempted by a higher priority thread.
- Fixed priorities – The thread always retains the priority it was initially assigned.

Real-time Class

The RT class has the following characteristics:

- Time sliced
- Fixed priorities



Dispatch Priorities

System threads are assigned a priority both within their class, and based on that priority, a system-wide dispatching priority.

Threads are assigned to a specific class, and without user intervention they remain in that class for the life of their process. Priority bands are assigned in a hierarchy based on class order, which cannot be changed.

If a new scheduling class is activated, such as real-time, it is inserted into its proper place in the hierarchy and all other existing classes are adjusted accordingly.



Sun Educational Services

Timesharing/Interactive Dispatch Parameter Table

globpri	quantm	ts_tqexp	ts_slpret	ts_mwait	ts_lwait
0	20	0	50	0	50
1	20	0	50	0	50
2	20	0	50	0	50
3	20	0	50	0	50
4	20	0	50	0	50
5	20	0	50	0	50
6	20	0	50	0	50
7	20	0	50	0	50
8	20	0	50	0	50
9	20	0	50	0	50
10	16	0	51	0	51
11	16	1	51	0	51
12	16	2	51	0	51
13	16	3	51	0	51
14	16	4	51	0	51
15	16	5	51	0	51
16	16	6	51	0	51
17	16	7	51	0	51
18	16	8	51	0	51
19	16	9	51	0	51
20	12	10	52	0	52
21	12	11	52	0	52
22	12	12	52	0	52
23	12	13	52	0	52
24	12	14	52	0	52
25	12	15	52	0	52
26	12	16	52	0	52
27	12	17	52	0	52
28	12	18	52	0	52
29	12	19	52	0	52

Timesharing/Interactive Dispatch Parameter Table

The process scheduler (or dispatcher) is the portion of the kernel that controls allocation of the CPU to processes. The timesharing dispatch parameter table consists of an array of entries, one for each priority. It is indexed by the thread's priority, and provides the scheduling parameters for the thread. Initially, it is assigned the amount of CPU time specified in quantum, in clock ticks.

If a process is using a lot of CPU time, it exceeds its assigned CPU time quantum (`ts_quantum`) and is given the new priority in `ts_tqexp`. The new priority is usually 10 less than its previous priority, but in return it gets a longer time slice.

A thread that blocks waiting for a system resource is assigned the priority `ts_slpret` when it returns to user mode. The value of `ts_slpret` in the 50s allows even the most I/O-bound job at least short access to the CPU after returning.



Sun Educational Services

Timesharing/Interactive Dispatch Parameter Table

globpri	quantm	tqexp	slpreat	maxwait	lwait
30	8	20	53	0	53
31	8	21	53	0	53
32	8	22	53	0	53
33	8	23	53	0	53
34	8	24	53	0	53
35	8	25	54	0	54
36	8	26	54	0	54
37	8	27	54	0	54
38	8	28	54	0	54
39	8	29	54	0	54
40	4	30	55	0	55
41	4	31	55	0	55
42	4	32	55	0	55
43	4	33	55	0	55
44	4	34	55	0	55
45	4	35	55	0	55
46	4	36	57	0	57
47	4	37	58	0	58
48	4	38	58	0	58
49	4	39	58	0	58
50	4	40	58	0	58
51	4	41	58	0	58
52	4	42	58	0	58
53	4	43	58	0	58
54	4	44	58	0	58
55	4	45	58	0	58
56	4	46	58	0	58
57	4	47	58	0	58
58	4	48	58	0	58
59	2	49	58	32000	58

Timesharing/Interactive Dispatch Parameter Table

CPU Starvation

If a thread does not get any CPU time in `ts_maxwait` seconds, its priority is raised to `ts_lwait`. (A value of 0 for `ts_maxwait` means 1 second.)

By default, the `ts_maxwait` time is 1 second. The new `ts_lwait` priority in the 50s should allow the thread to get some amount of CPU time if any timesharing threads are able to run at all.

The `dispadmin` command is used to make temporary changes to the timesharing/interactive dispatch table. It is discussed later in this module. See the man page for `ts_dptbl(4)` to make permanent changes to the table.

A different table is used for the Enterprise 10000. It is loaded from `/platform/sun4u1/kernel/sched`. Use the `dispadmin` command to view it.



Dispatch Parameter Table Issues

For time-sharing class processes:

- Reducing time quanta favors interactive processes
- Raising time quanta favors compute-bound and large processes
- Using the `ts_maxwait` and `ts_lwait` fields controls CPU starvation
- Slightly raising the values of `ts_tqexp` causes the priority of compute-bound processes to drop more slowly
- Changing the table can be done to fit your workload

Dispatch Parameter Table Issues

When you change the size of the time quantum for a particular priority, you change the behavior of the CPU that it is using. For example, by increasing the size of the quantum, you help ensure that data the thread is using remains in the CPU caches, providing better performance. By decreasing the size of the quantum, you favor I/O-bound work that gets on and off of the CPU quickly.

With very heavy CPU utilization, the CPU starvation fields can ensure that all threads get some CPU time. Generally, if you are relying on starvation processing, you need more CPUs for the system.

If your workload changes slowly between I/O bound and CPU bound, you can alter the `ts_tqexp` and `ts_slpresp` fields to raise and lower priorities more slowly.



Sun Educational Services

Timesharing Dispatch Parameter Table (Batch)

globpri	quanta	tqexp	slpret	nswait	lwait
0	100	0	10	5	10
1	100	0	11	5	11
2	100	1	12	5	12
3	100	1	13	5	13
4	100	2	14	5	14
5	100	2	15	5	15
6	100	3	16	5	16
7	100	3	17	5	17
8	100	4	18	5	18
9	100	4	19	5	19
10	80	5	20	5	20
11	80	5	21	5	21
12	80	6	22	5	22
13	80	6	23	5	23
14	80	7	24	5	24
15	80	7	25	5	25
16	80	8	26	5	26
17	80	8	27	5	27
18	80	9	28	5	28
19	80	9	29	5	29
20	80	10	30	5	30
21	80	11	31	5	31
22	80	12	32	5	32
23	80	13	33	5	33
24	80	14	34	5	34
25	80	15	35	5	35
26	80	16	36	5	36
27	80	17	37	5	37
28	80	18	38	5	38
29	80	19	39	5	39

Timesharing Dispatch Parameter Table (Batch)

The table shown in the overhead image is from the SunOS 5.2 operating system, where the table favored batch work. Note that the quanta are five times larger, and priorities drift up and down more slowly as the quantum is or is not used up.

The table allows batch work to use the CPU caches, such as the page descriptor cache (PDC) and level one and level two caches, more effectively. It also forces the thread to "earn" its way to I/O-bound status, as the thread works its way up over several dispatches. Using the `nice` and `priocntl` commands may change the calculated priorities for a thread.

A hybrid table could be built, using the upper half of the current table and the lower half of this table, which allows I/O-bound work to move up quickly, and CPU-bound work to use the hardware more efficiently.

globpri	quantum	tqexp	slpref	maxwait	lwait
30	40	20	40	5	40
31	40	21	41	5	41
32	40	22	42	5	42
33	40	23	43	5	43
34	40	24	44	5	44
35	40	25	45	5	45
36	40	26	46	5	46
37	40	27	47	5	47
38	40	28	48	5	48
39	40	29	49	5	49
40	20	30	50	5	50
41	20	31	50	5	50
42	20	32	51	5	51
43	20	33	51	5	51
44	20	34	52	5	52
45	20	35	52	5	52
46	20	36	53	5	53
47	20	37	53	5	53
48	20	38	54	5	54
49	20	39	54	5	54
50	10	40	55	5	55
51	10	41	55	5	55
52	10	42	56	5	56
53	10	43	56	5	56
54	10	44	57	5	57
55	10	45	57	5	57
56	10	46	58	5	58
57	10	47	58	5	58
58	10	48	58	5	59
59	10	49	58	32000	58

Timesharing Dispatch Parameter Table (Batch)

The remainder of the batch dispatch parameter table is shown in the above overhead.



The dispadmin Command

- Displays or changes scheduler parameters
- Uses options:
 - -l – List available scheduling classes
 - -c class – Specifies the class whose parameters are to be displayed or changed
 - -g – Displays configured parameters
- Provides a simple way of formatting control file
 - -s file – Sets parameters from a file

The dispadmin Command

The dispatcher administration command `dispadmin` enables the system administrator to inspect and change the dispatch parameter table for the timesharing or real-time classes. (The interactive class uses the timesharing (TS) table, and the system class has only a list of priorities.)

This command can provide a list of the active, loaded classes (-l), or allow you to inspect or change the current table. For example, to change the TS table, you would:

1. Save the current table to a file.

```
dispadmin -c TS -g > tfile
```

2. Edit `tfile` to change the table.

3. Make the changed table active for the life of the boot by running:

```
dispadmin -c TS -s tfile
```



dispadmin Example

```
* dispadmin -c TS -g
#Time Sharing Dispatcher Configuration
RES=1000

*ts_quantum ts_bqexp ts_sipret ts_maxwait ts_lwait PRIORITY LEVEL
200      0      50      0      50      # 0
200      0      50      0      50      # 1
200      0      50      0      50      # 2
200      0      50      0      50      # 3
200      0      50      0      50      # 4
200      0      50      0      50      # 5
200      0      50      0      50      # 6
...
40      44      58      0      59      # 54
40      45      58      0      59      # 55
40      46      58      0      59      # 56
40      47      58      0      59      # 57
40      48      58      0      59      # 58
20      49      59      32000    59      # 59
```

dispadmin Example

The above overhead shows how to inspect a dispatch table.

The resolution for the quanta defaults to milliseconds, which is the inverse of the resolution field (RES = 1000). A different resolution can be specified by using the **-r** option. If you want the resolution to be displayed in hundredths of a second, use the command:

```
dispadmin -r 100 -c TS -g
```

If you change the table, be sure to set the quanta in units that correspond to the RES value.

Note – See the **ts_dptbl(4)** man page for instructions on how to make permanent changes to the table.



The Interactive Scheduling Class

- Is used to enhance interactive performance
- Is the default scheduling class for processes in Common Desktop Environment and OpenWindows™ sessions
- Uses most of the time-sharing class facilities
- Boosts the priority of the task in the active window by 10 points
 - Priority is reset when it is no longer the active window.
- Does not boost processes changed using nice or other commands

The Interactive Scheduling Class

The interactive scheduling class is derived from the timesharing class, using most of its code and data structures. It is used to enhance interactive performance.

It is the default scheduling class for processes running in OpenWindows or CDE. It automatically adds 10 to the calculated priority for the task in focus; that is, the one running in the active window. This gives better performance to the job a user is most likely to be waiting for.

When the active window changes, the priority boost is transferred to the new focus window.

If you adjust the in-focus process' priority by nice or priocntl, the boost is not applied.



Real-Time Scheduling

- Real-time systems have rigid response time or dispatch latency requirements.
- Failure to meet the time constraints can cause problems.
- General purpose systems have problems with real time.
 - Page faults have an unknown time to resolve.
 - All pages are locked in memory.
 - They include shared libraries and shared memory.
 - A non-preemptable kernel can cause potential CPU delays.
 - A preemptable kernel is used.

Real-Time Scheduling

Hard real-time systems must respond to an interrupt or other external event within a certain period of time (the *latency*), or problems arise. While SunOS does not provide a hard real-time facility, it does provide soft real-time capability, where the system tries to run the real-time thread as soon as possible, typically in less than a millisecond.

There have been problems with real-time, general-purpose systems in the past, which the Solaris real-time facility tries to avoid, as shown in the overhead image. The problem area of page faults are avoided by locking all possible pages that the process could use in memory. To avoid delays from lower priority jobs using the CPU, a thread can be interrupted at any time in favor of a real-time thread.

Real-time work can have a significant impact on the system and should be used sparingly: it has higher priority than the system threads. Never put a general purpose program, such as a DBMS, into the real-time class to try to improve its performance.



Real-Time Issues

- Real-time processes can hurt the system if they:
 - Use lots of memory – all pages are locked in memory
 - Are compute-intensive – other tasks may be locked out and system daemons may suffer
 - fork other processes – children inherit real-time
 - Get stuck in a loop – may be hard to stop
- Make sure that you *really* need real time before using it.
 - Consider using a real-time thread instead of a whole process
 - Set `hires_tick` to 1 to set clock interrupts to 1000

Real-Time Issues

Threads running in real time have an impact on the rest of the processes competing for system resources, especially if they are:

- Large – All of their pages are locked into memory, reserving potentially large amounts of main storage.
- Compute intensive – They will prevent system threads from accessing the CPU if they use too much CPU time.
- fork other processes – Child processes inherit their scheduling class from their parent unless `priocntl` is used.
- Stuck in a loop – It can be difficult to stop a runaway real-time thread on a small system since timesharing threads may be locked out.

If you do real-time processing:

- Remember, real-time threads do not change priorities like timesharing threads. Their priority is fixed for the duration.
- Consider using application threads. A bound thread with its LWP set into the real-time class by the `priocntl` system call will have less impact on the system, and can segment real-time and normal work more easily.
- If you are concerned about real-time dispatch latency, set `hires_tick` to 1 in `/etc/system`. This will cause the system clock to run at 1000 Hz instead of 100 Hz. It will cause more overhead on the CPU processing clock interrupts, but real-time work will be dispatched more quickly.



Sun Educational Services

Real-Time Dispatch Parameter Table

g1dpc1	quantum	g1objc1	quantum
100	100	130	40
101	100	131	40
102	100	132	40
103	100	133	40
104	100	134	40
105	100	135	40
106	100	136	40
107	100	137	40
108	100	138	40
109	100	139	40
110	60	140	20
111	60	141	20
112	60	142	20
113	60	143	20
114	60	144	20
115	60	145	20
116	60	146	20
117	60	147	20
118	60	148	20
119	60	149	20
120	60	150	10
121	60	151	10
122	60	152	10
123	60	153	10
124	60	154	10
125	60	155	10
126	60	156	10
127	60	157	10
128	60	158	10
129	60	159	10

Real-Time Dispatch Parameter Table

The above overhead lists the default values for the real-time dispatch parameter table. When a thread is put into the real-time class by `priocntl0`, the priority and the time quantum can be specified at that time. No thread or process is put into the real-time class by default.

Since the real-time priorities are fixed, the time quantum and global priority do not change for the life of the thread unless they are modified using `priocntl1`. This also means that the fields from the timesharing dispatch parameter table that manage the priority changes are not needed.

Note – For instructions on how to make permanent changes to this table, see the man page for `rt_dptbl(4)`.



The priocntl Command

- Works only at the process level
- Uses options:
 - -l – List currently loaded scheduling classes
 - -d – Display a process' scheduling parameters
 - -s – Set a process' scheduling parameters
 - -e [-c class] – Create a new process
- Makes the priocntl(2) system call available to programs
 - Works on threads and LWPs

The priocntl Command

The priocntl command is used to change the scheduling behavior of a process, or, by using the priocntl(2) system call, an individual LWP.

Using the priocntl command or call, you can change:

- Scheduling class
- Relative priority within the class
- Time quantum size

It can also be used to display current configuration information for the system's process scheduler or execute a command with specified scheduling parameters.

You must be the root user to place a process into the real-time class.



The priocntl Command

Example

```
# priocntl -e -c RT -p 50 -t 500 command args
```

- Run *command* with operand *args*
- Use the real-time scheduling class
- Set real-time class relative priority to 50
 - Global priority is 150 (50 + base real-time class priority).
 - Use a time quantum of 500 milliseconds

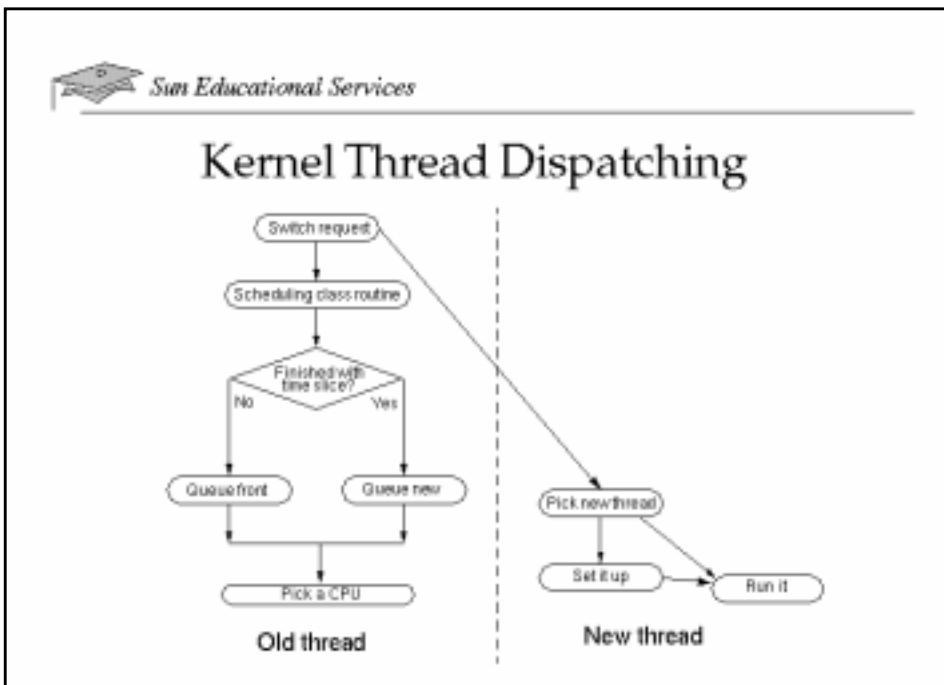
The priocntl Command

For example, the *priocntl* command allows a process to be placed in real time with a specified priority and time quantum.

The command shown above runs *command* at a global real-time priority of 150 (response time [RT] relative priority 50 + the base priority of the RT class) and with a time quantum of 500 milliseconds.

You can specify an infinite time quantum with the *priocntl(2)* system call by using the special value *RT_TQINF* for the time quantum.

Note – In order to change the scheduling parameters of a process using *priocntl*, the real or effective user ID of the user invoking *priocntl()* must match the real or effective user ID of the receiving process, or the effective user ID of the user must be superuser.



Kernel Thread Dispatching

The overhead image above shows a simplified flow chart of the operation of kernel thread dispatching.

Old Thread

The switch request (pre-empt) routine is called either when a higher priority thread is put on the dispatch queue or a thread has used up its time slice. It has two responsibilities: remove the current thread from the CPU, and assign a new thread in its place.

The first call is made to a scheduling class routine, it allows the scheduling class to perform any necessary work. These routines check to see if the departing thread should go to the front or the back of the dispatch queue. A thread goes to the front of the dispatch queue if it has not used up its time slice.

Since each CPU has its own dispatch queue, it is necessary to determine which CPU's queue the thread should be placed on. For a timesharing thread, this choice is made as follows:

- If a thread has been out of execution for less than three ticks, it is placed on the dispatch queue of the CPU it last executed on.
- Otherwise, the CPU that is currently running the lowest priority thread is chosen.

For a real-time thread, the dispatch queue of the CPU running the lowest priority thread is selected.

New Thread

After the replaced thread has been placed on a dispatch queue, a new thread must be chosen to run on the CPU. The priority is as follows (1 being the highest and 5 the lowest):

1. Real-time threads on the current CPU's dispatch queue
2. Real-time threads on any CPU's dispatch queue
3. Other threads on the current dispatch queue
4. Other threads on any dispatch queue
5. The idle thread (priority -1)

The chosen thread is then loaded onto the CPU (for example, its registers restored), and run. If the thread selected is the same thread that was just taken off of the CPU, it is run since it is already loaded on the CPU.



Processor Sets

- Allows exclusive use of groups of processors by certain processes
 - Also known as *CPU fencing*
- Is very different from the pbind(1M)
- Is managed by the psrset(1M) command
- Is controlled by the root user
- Has system-defined processor sets which can be used by any user
- Forces DR to release bindings if necessary

Processor Sets

Processor sets are a feature provided by the SunOS 5.6 operating system that add the ability to "fence" or isolate groups of CPUs for use by specially designated processes. This allows these processes to have guaranteed access to CPUs that other processes, including the system itself, cannot use. A processor may belong to only one processor set at a time.

Processor sets differ significantly from processor binding (which is done using the pbind command or the `processor_bind(2)` system call). The pbind command designates a single specific processor that the process must run on, but other processes can continue to use that processor. With processor sets, the CPUs are dedicated to the processes that are assigned to the set.

Work that is not assigned to a specific processor set must be run on the CPUs that are not part of a processor set. You must ensure that you have enough CPU capacity available for this non-associated work.

System CPUs can be grouped into one or more processor sets by the `psrset -c` command. These processors will remain idle until processes (technically, LWPs) are assigned to them by the `psrset -b` command. Processors can be added and removed from a processor set at any time with the `psrset -a` and `psrset -r` commands, respectively. Processor set definitions can be viewed with the `psrset -p` command.

Dynamic Reconfiguration Interaction

If a processor set is completely removed from the system by dynamic reconfiguration (DR) detach, the processes using the processor set will be unbound, and able to run on any CPU in the system. If a large part of a processor group is removed, you might need to use `psrset -a` to add additional processors to the processor set.

The system does not retain processor set configuration across DR operations. If you use DR Attach to connect a board to a system in one or more processor sets, you must use `psrset -a` to add the CPUs back to that set. System-defined processor sets will be automatically restored.



Using Processor Sets

- You must leave one processor for the system.
 - Make sure you leave enough CPU capacity for non-fenced work.
- A CPU may be in only one processor set.
- The system will not adjust dynamically.
 - The tools do not show processor set queues.
 - Queueing on a processor set could be missed.
- You can change CPU membership dynamically.
 - You need to watch fenced processes carefully.

Using Processor Sets

Once a processor set has been created, processes are assigned to it by using the `psrset -b` command. These processes may run only on the processors in the specified processor set, but they can run on any processor in the set. Any child processes or new LWPs created by a process in a processor set will be set to run in the same processor set. Processes are unbound from a processor set with the `psrset -u` command.

Only the root user may create, manage, and assign processes to these processor sets.

The `psrset -q` command displays the processes bound to each processor set, and the `psrset -i` command shows the characteristics of each processor set.

As well as manually created processor sets, the system might also provide automatically defined system processor sets. These processor set definitions cannot be modified or deleted. Any user can assign processes to system-defined processor sets, but other processes will not be excluded from these CPUs. A processor can be in a system or non-system processor set, but not both at the same time.

Processor sets can also be managed from within a program by using the `psrset(2)` system call.



The Run Queue

- A count of kernel threads waiting to run is kept.
 - It contains a total of all system dispatch queues.
 - It is not scaled by the number of CPUs.
- A depth of 3–5 per CPU is usually OK.
 - This depends on the type of work being run.
- There is no way to tell what is waiting or how long it has been waiting.

The Run Queue

The performance monitoring tools report the length of the CPU run queue. The run queue is the number of kernel threads waiting to run on all system dispatch queues. It is *not* scaled by the number of CPUs. A depth of three to five is normally acceptable, but if heavily CPU-bound work is constantly being run, this may be too high.

It is not possible to tell what work is being delayed nor how long it has run. Remember, higher priority work can constantly preempt lower priority work, subject to any CPU starvation processing done by the thread's scheduling class.



CPU Activity

- User – The user process is running.
- System – Kernel is running.
 - This includes system thread time and user system calls.
 - Wait I/O – The CPUs are idle, but a disk device is active.
 - Idle – The system is waiting.
 - Some reports add Wait I/O and Idle.
 - This is usually reported as Idle time.
 - You can check the tool's man page.

CPU Activity

The performance monitoring tools report four different kinds of CPU activity. These types are reported as percentages.

- User – Programming in the user process is being run. This is the amount of time the CPU spends running the user's program. This time is under the programmer's control. This does not include system calls, but does include shared library routines.
- System – System time is reported when programming resident in the kernel is being run. This is the amount of time the CPU spends executing kernel code on behalf of the user program. This code can be executed due to interrupts, system thread dispatches, and application system calls.
- Wait I/O – All CPUs are idle, but at least one disk device is active. This is the amount of time spent waiting for blocked I/O operations to complete. If this is a significant portion of the total, the system is I/O bound.

- Idle – None of the CPUs are active, and no disk devices are active. If CPU spends a lot of time idle, a faster processor will only increase idle time.

If the CPU is not spending a lot of time idle (less than 15 percent), then threads which are runnable have a higher likelihood of being forced to wait before being put into execution. In general, if the CPU is spending more than 70 percent in user mode, the application load might need some balancing. In most cases, 30 percent is a good high-water mark for the amount of time that should be spent in system mode.

Some reporting tools, such as `vmstat`, add Wait I/O and Idle time together, usually reporting it as Idle time.

To locate the device(s) causing a high Wait I/O percentage, check the `%w` column, reported by `iostat -x`. The amount of CPU time used by an individual process can be determined by using `/usr/ucb/ps -au` or system accounting.



Sun Educational Services

CPU Performance Statistics

Activity	Origin	vmstat (Unit)	sar (Unit)
Time CPU is executing in user mode	cpu.sysinfo[CPU_USER]	us (%)	%usr (%)
Time CPU is executing in kernel mode	cpu.sysinfo[CPU_KERNEL]	sy (%)	%sys (%)
Time CPU is waiting for I/O	cpu.sysinfo[CPU_WAIT]	sw (number of)	%wio (%)
Time CPU is idle	cpu.sysinfo[CPU_IDLE]	id (number of)	%idle (%)
Average number of threads blocked on I/O	sysinfo.waiting	b (number of)	
Average number of runnable threads over the interval	sysinfo.runque	r (number of)	runq-av (number of)
Percentage of time the run queue was empty	sysinfo.runqcc		%runqcc (%)
Average number of thread context switches	cpu.sysinfo.pswtch	cs (number per second)	pswch/s (number per second)

CPU Performance Statistics

The table in the above overhead compares the data available from `vmstat` and `sar` relevant to CPU activity. Sample outputs of the `sar` command with CPU-related options are shown. The `-u` option reports CPU utilization.

```
# sar -u
```

```
SunOS rafael 5.7 Generic sun4u      06/10/99

00:00:00    %usr    %sys    %wio    %idle
01:00:00      0        0        0     100
02:00:00      0        0        0     100
03:00:00      0        0        0      99
...
...
14:20:00      4        2        0      94
14:40:02      3        2        8      88
Average       1        0        1      98
#
```

The fields in this output are:

- %usr – Lists the percentage of time that the processor is in user mode.
- %sys – Lists the percentage of time that the processor is in system mode.
- %wio – Lists the percentage of time the processor is idle and waiting for I/O completion.
- %idle – Lists the percentage of time the processor is idle and is not waiting for I/O.

The `sar -q` command reports average run queue length while occupied, and the percent of time the queue is occupied. Sample output is as follows.

```
# sar -q
SunOS rafael 5.7 Generic sun4u      06/10/99

00:00:00 runq-sz %runocc swpq-sz %swpocc
01:00:00      1.0      0
02:00:00
03:00:00      1.6      0
...
...
14:20:00      1.7      2
14:40:02      2.4      2

Average      1.8      1
#
```

The fields in this output are:

- runq-sz – The average number of kernel threads in memory waiting for a CPU to run. Typically, this value should be less than 2. Consistently higher values mean that the system may be CPU bound.
- %runocc – The percentage of time a thread was on any dispatch queue.

The `sar -w` command reports system swapping and switching activity. Sample output is shown.

```
# sar -w

SunOS rafael 5.7 Generic sun4u      06/10/99

00:00:00 swpin/s bswin/s swpot/s bswot/s pswch/s
01:00:00    0.00    0.0    0.00    0.0     176
02:00:00    0.00    0.0    0.00    0.0     175
...
...
14:20:00    0.00    0.0    0.00    0.0     274
14:40:02    0.00    0.0    0.00    0.0     259

Average    0.00    0.0    0.00    0.0     205
#
```

The applicable field in this output is:

- `pswch/s` – The number of kernel thread switches per second.

The `vmstat` command also reports data relevant to CPU activity. Sample output of the `vmstat` command is shown.

```
# vmstat 2

procs      memory          page          disk          faults          cpu
r b w  swap   free   re   mf pi po fr de sr f0 s0 s6 --   in   sy   cs us sy id
0 0 0 30872 3688   0   2   8 13 24   0   5   0   1   0   0   184   370   147   1   0   98
0 0 0 164696 976    0   3   0 40 124   0 102   0   1   0   0   175   430   170   1   0   98
0 0 0 164696 976    0   1 12   0   0   0   0   0   1   0   0   314   766   203   2   2   96
1 0 0 164592 840    1 289   4 40 140   0 93   0   1   0   0   244   5362   250   9   14   77
1 0 0 164464 984    0 1098   0   0   0   0   0   0   0   0   0   291 20707   282   33   67   0
1 0 0 164464 992    0 1190   16   0   0   0   0   0   2   0   0   385 21714   332   32   68   0
1 0 0 164512 960    0 1151   116 48 108   0 115   0 16   0   0   388 20748   360   30   70   0
1 0 0 164520 976    0 1161   8   0   0   0   0   0   1   0   0   174 21588   288   27   73   0
1 0 0 164512 952    0 1160   4   0   0   0   0   0   0   0   0   329 21575   312   32   68   0
1 0 0 164512 928    0 1160   8   4   4   0   0   0   1   0   0   356 21156   344   31   69   0
0 0 0 164504 920    0 1098   0   0   0   0   0   0   0   0   0   201 19975   263   38   62   0
0 0 0 164720 1120   0   3   4   0   0   0   0   0   1   0   0   182 1052    186   2   1   96
0 0 0 164712 1112   0   0   0   0   0   0   0   0   0   0   0   182 611    191   0   0   99
0 0 0 164712 1104   0   0   4   0   0   0   0   0   1   0   0   174 489    181   1   1   98
0 0 0 164712 1096   0   16   8   0   0   0   0   0   21   0   0   266 589    204   1   2   97
0 0 0 164720 1096   0   0   0   0   0   0   0   0   0   0   0   276 548    187   2   0   98
0 0 0 164728 960    0   34 100   0   8   0   6   0 13   0   0   298 668    270   0   4   96
^C#
```

The fields in this output are:

- us – Percentage of user time.
- sy – Percentage of system time.
- id – Percentage of idle time.
- r – The number of kernel threads in the dispatch (run) queue; the number of runnable processes that are not running.
- b – The number of blocked kernel threads waiting for resources, I/O, paging, and so forth; the number of processes waiting for a block device transfer (disk I/O) to complete.
- cs – CPU context switch rate.



Available CPU Performance Data

- `cpu_sysinfo.cpu[CPU_USER]` – User mode CPU
- `cpu_sysinfo.cpu[CPU_KERNEL]` – Kernel mode CPU
- `cpu_sysinfo.cpu[CPU_WAIT]` – CPU idle, block I/O active
- `cpu_sysinfo.cpu[CPU_IDLE]` – CPU idle
- `cpu_sysinfo.pswitch` – Number of thread switches
- `sysinfo.runque` – Average number of runnable threads
- `sysinfo.runocc` – Percentage of time a thread was on any dispatch queue

Available CPU Performance Data

The above overhead image shows the origin and meaning of the data reported in the CPU monitoring reports.



CPU Control and Monitoring

Processor control and information is reported by:

- mpstat – Displays CPU usage statistics
- psradm – Enables and disables individual CPU
- psrinfo – Determines which CPUs are enabled
- prtconf – Shows device configuration
- prtdiag – Prints system configuration and diagnostic information
- Process Manager – Shows current activity
- psrset – Manages processor sets

CPU Control and Monitoring

There are a number of commands that provide CPU control and status information:

- mpstat – Show individual CPU activity statistics
- psrinfo – Display which CPUs are enabled and their speeds
- psradm – Enable/disable CPUs
- /usr/platform/sun4X/sbin/prtdiag – Print system hardware configuration and diagnostic information
- prtconf – Show device configuration
- sysdef – Show software configuration
- psrset – Manage processor sets



The mpstat Command

- Displays per-processor performance statistics
- Is similar to vmstat in operation
- Is used to check per-CPU loads and load balancing
- Is usually not needed in normal tuning
 - It can be useful with erratic performance.
- Uses lockstat to resolve locking problems

The mpstat Command

The mpstat performance monitor is used to display per-processor performance statistics, in a manner similar to vmstat. It can be used to check per-CPU load/load balancing behavior.

This data can be used to detect load imbalances between processors. Usually, mpstat is not needed for normal tuning tasks, but is very helpful when the cause of bad performance, especially if it is CPU related, cannot be located.

It can also point to locking problems, which can be identified with the lockstat command.

Sample output of the mpstat command is as follows.

```
# mpstat 2
CPU minf mjf xcal  intr ithr  csw icsw migr smtx  srw syscl  usr sys  wt idl
 0   0   0   0     0   0   35   0    2   1     0   96   0   0   0   100
 2   0   0   4   208   8   34   0    2   1     0   81   0   0   0   100
CPU minf mjf xcal  intr ithr  csw icsw migr smtx  srw syscl  usr sys  wt idl
 0   2   0   0     0   0   30   0    1   0     0   54   0   0   0   100
 2   0   0   4   212  11   21   0    1   0     0   25   0   0   0   100
CPU minf mjf xcal  intr ithr  csw icsw migr smtx  srw syscl  usr sys  wt idl
 0   0   0   0     0   0   31   0    1   0     0   53   0   0   2   98
^C#
```

The output contains the following fields:

- CPU – Processor ID
- minf – Minor faults
- mjf – Major faults
- xcal – Inter-processor cross-calls (which occur when one CPU wakes up another CPU by interrupting it)
- intr – Interrupts
- ithr – Interrupts as threads (not counting clock interrupt)
- csw – Context switches
- icsw – Involuntary context switches
- migr – Thread migrations (to another processor)
- smtx – Spins on mutexes (mutual exclusion lock), locks not acquired on first try, and number of times the CPU failed to obtain a mutex immediately
- srw – Spins on readers/writer locks (lock not acquired on first try)
- syscl – System calls
- usr – Percent user time
- sys – Percent system time
- wt – Percent wait time
- idl – Percent idle time



mpstat Data

- xcal – Number of interprocessor cross calls (`cpu_sysinfo.xcalls`)
- intr – Number of interrupts received on each CPU (`cpu_sysinfo.intr`)
- migr – Number of threads which migrated from another CPU (`cpu_sysinfo.cpumigrate`)
- smtx – Number of failed adaptive mutex_enter () calls (`cpu_sysinfo.mutex_adenters`)
- srw – Number of times readers/writer locks were not acquired on the first try (`cpu_sysinfo.rw_wrfails`, `cpu_sysinfo.rw_rdfails`)

mpstat Data

The above overhead image shows the data reported by `mpstat` and its origin. `mpstat` also provides CPU utilization data for each CPU.



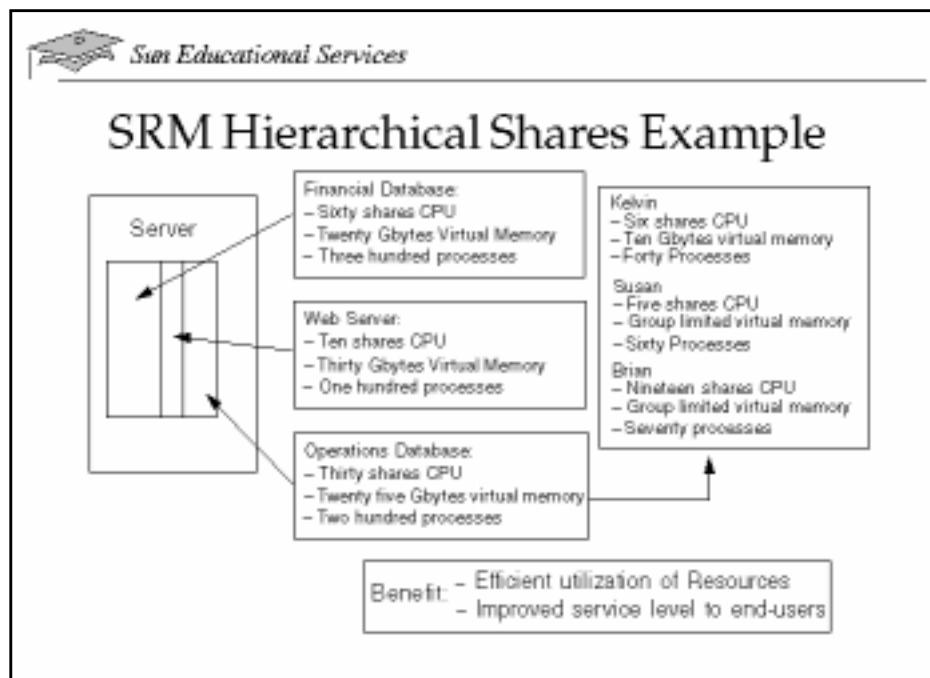
Solaris Resource Manager

- Is Sun's software tool for allocating system resources to users, groups, and enterprise applications
- Offers the ability to control and allocate:
 - CPU time
 - Processes
 - Virtual memory
 - Connect time
 - Logins
- Allows IT managers to consolidate applications on a single UNIX® server

Solaris Resource Manager

Solaris Resource Manager (SRM) is Sun's software tool for making resources available to users, groups, and enterprise applications. SRM offers the ability to control and allocate CPU time, processes, virtual memory, connect time, and logins.

SRM allocates resources according to a defined policy and prevents resource allocation to users and applications that are not entitled to them. Once a resource policy is set, mission-critical applications will get the resources they demand. SRM prevents runaway processes from consuming all of the available CPU power or virtual memory.



SRM Hierarchical Shares Example

CPU time is allocated based on an assigned number of shares, rather than on a flat percentage basis. When applications are idle or are consuming less than their designated CPU allotment, other applications can take advantage of the unused processing power. The granularity of control can be as deep or flat as the system administrator desires. Shares can be allotted to a department, and then to individuals in the department.

In the above overhead, the system administrator divides system resources between three groups of users: Financial Database (60 shares), Web Server (10 shares), and Operations Department (30 shares). Resources allocated to the Operations Department are then further distributed to Kelvin (6 shares), Susan (5 shares), and Brian (19 shares). The 30 shares allocated for CPU resources to the Operations Department represents a minimum of 30 percent (30 of 100 shares) if the Financial Database and Web Server are also in operation. If the Financial Database was not running, Operations would receive 75 percent of CPU resources (30 of 40 shares). The ratio of shares is important, not the actual number of shares.



SRM Limit Node

- Defines a resource control policy for users or groups of users rather than a system-wide policy
- Has as a fundamental building block in SRM, the lnode (limit node)
- Has an lnode for each user and group of users
- Uses an lnode which contains attributes that correspond to a resource control policy
- Has the kernel access lnodes to determine a user's resource limits and resource usage

SRM Limit Node

By default, the Solaris operating environment includes system parameters that impose limits on resources; for example, the number of processes per user. However, these system parameters are defined within the kernel for all users, and do not allow different values for different users. SRM defines a resource control policy for each individual user or for a group of users. Thus the SRM system administrator has a great deal of flexibility in defining the resource control policy for a system.

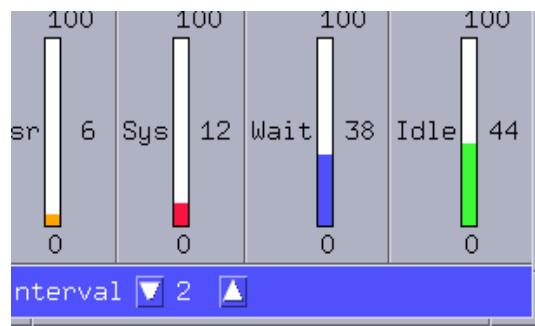
The fundamental building block in SRM is a per-user persistent object called an lnode (limit node). An lnode exists for every user and group of users on the system. The lnode object contains system and user attributes that correspond to the resource control policy for that user. The kernel accesses lnodes to determine a user's specific resource limits and to track that user's resource usage. When a user starts a process, the user's lnode records the process and its activity. As the process runs and requests resources, the kernel determines whether or not the process is entitled to the requested resources based on the lnode information.

CPU Monitoring Using SE Toolkit Programs

cpu_meter.se

The `cpu_meter.se` program is a basic meter which shows usr, sys, wait, and idle CPU as separate bars.

```
# /opt/RICHPS/se /opt/RICHPS/examples/cpu_meter.se
```



vmmonitor.se

The `vmmonitor.se` program is an old and oversimplified `vmstat`-based monitor script that looks for low RAM and swap space.

mpvmstat.se

The `mpvmstat.se` program provides a modified `vmstat`-like display. It prints one line per CPU.

Check Your Progress

Before continuing on to the next module, check that you are able to accomplish or answer the following:

- Describe the Solaris scheduling classes
- Explain the use of dispatch priorities
- Describe the time-sharing scheduling class
- Explain Solaris real-time processing
- List the commands that manage scheduling
- Describe the purpose and use of processor sets
- Introduce Solaris Resource Manager

Think Beyond

Consider the following questions:

- Why are there four scheduling classes? What was the point of adding the interactive scheduling class?
- Why would you want to change a dispatch parameter table?
- When might you want to use the SunOS operating system's real-time capability?
- What kinds of work might you use processor sets for?

Objectives

Upon completion of this lab, you should be able to:

- Read a dispatch parameter table
- Change a dispatch parameter table
- Examine the behavior of CPU-bound and I/O-bound processes, both individually and together
- Run and observe a process in the real-time scheduling class

Tasks

CPU Dispatching

Complete the following steps:

1. Display the configured scheduling classes by typing:

```
dispadmin -l
```

What classes are displayed?

2. Use dispadmin to display the timesharing dispatch parameter table by typing:

```
dispadmin -c TS -g
```

What is the longest time quantum for a timeshare thread?

3. Use the dispadmin command to display the timeshare dispatch parameter table so that the time quantums are displayed in clock ticks. What command did you use?

4. Display the real-time dispatch parameter table by typing:

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

What is the smallest time quantum for a real-time thread?

5. Display the configured scheduling classes again. What has changed?

Process Priorities

Recall how process priorities get assigned:

- A process gets assigned a priority within its class (timeshare process priorities range from 0 through 59 with 59 being the highest priority).
- Each priority has an associated *quantum*, which is the amount of CPU time allocated in ticks.
- If a process exceeds its assigned *quantum* before it has finished executing, it is assigned a new priority, *tqexp*.
- The new priority, *tqexp*, is lower but the associated *quantum* is higher so the process gets more time.
- If a process (or thread) blocks waiting for I/O, it is assigned a new priority, *slpret*, when it returns to user mode.
- *slpret* is a higher priority (in the 50s) allowing faster access to the CPU.
- If a process (or thread) does not use its time quantum in *maxwait* seconds (*maxwait* of 0 equals 1 second), its priority is raised to *lwait* which is higher and assures the thread will get some CPU time.

The purpose of this portion of the lab is to observe the process priorities during execution of a program. The two programs that you will be executing are `dhystone` and `io_bound`. `dhystone` is an integer benchmark that is widely used for benchmarking UNIX systems. `dhystone` provides an estimate of the raw performance of processors. (You may also be familiar with a floating-point benchmark called `whetstone` which was popular for benchmarking FORTRAN programs.) `dhystone` is entirely CPU bound, and runs blindingly fast on machines with high-speed caches. Although it is good measure for programs that spend most of their time in some inner loop, it is a poor benchmark for I/O-bound applications.

The `io_bound` program, as the name implies, is an I/O-intensive application. I/O-bound processes spend most of their time waiting for I/O to complete. Whenever an I/O-bound process wants CPU, it should be given CPU immediately. The scheduling scheme gives the I/O-bound processes higher priority for dispatching.

Complete the following steps, as root, using the lab4 directory.

1. Run the dhystone command. dhystone without any parameters makes 500000 passes. If this executes too fast to monitor, pass a parameter to it such as `./dhystone 1000000`.

```
% ./dhystone
```

2. Use `ps -cle | grep dhry` in another terminal window to view dhystone process's priority.

What priority has it been assigned?

Why is it so low? (Get several "snapshots" of the process priority by using `ps -cle`.)

3. When dhystone has completed, type:

```
./io_bound
```

This runs an I/O-intensive application.

4. Type `ps -cle` in another terminal window to view the priority of this process.

What priority has it been assigned?

Why? (Again, get several "snapshots" of the priority using `ps -cle`.)

5. Type:

```
/usr/proc/bin/ptime ./io_bound
```

-
-
6. Record the results.

7. Type:

```
/usr/proc/bin/ptime ./dhystone
```

8. Record the results.

9. Type:

```
./test.sh
```

This shell script runs both `dhystone` and `io_bound` and stores the results in two files, `io_bound.out` and `dhystone.out`. It then waits for both of them to complete. Record the results.

- ▼ `io_bound` results

- ▼ `dhystone` results

Why did the `dhystone` process take longer to execute when it was run in conjunction with the `io_bound` process?

The Dispatch Parameter Table

Using the lab4 directory, as root perform the following steps:

1. Type:

```
dispadmin -g -c TS > ts_config
```

This saves the original timeshare dispatch parameter table.

2. Type:

```
dispadmin -s ts_config.new -c TS
```

You have just installed a new timeshare dispatch parameter table.
Do not look at the new table at this time.

3. Type:

```
./test.sh
```

Note the results. Given the results, can you make an educated guess as to the nature of the ts_config.new dispatch parameter table?

Look at ts_config.new to confirm.

4. Type:

```
ps -cle
```

Do you notice anything strange about some of the priorities you see? Which processes have very low priorities? Why?

5. Type:

```
dispadmin -s ts_config -c TS
```

This restores the original table.

Real-Time Scheduling

Run this exercise as `root` from the `lab4` directory. Use the following steps:

1. Run `test.sh` as a real-time process. Type:

```
priocntl -e -c RT ./test.sh.
```

2. Record the results for `io_bound` and `dhrystone`. What is different from the timeshare case?

Can you explain this?

Note – Running these processes as real time might cause your system to temporarily “hang.”

▼ `io_bound` results

▼ `dhrystone` results

Objectives

Upon completion of this module, you should be able to:

- Describe the purpose of a cache
- Describe the basic operation of a cache
- List the various types of caches implemented in Sun systems
- Identify the important performance issues for a cache
- Identify the tuning points and parameters for a cache
- Understand the memory system's hierarchy of caches and their relative performances

Relevance

Discussion – The following questions are relevant to understanding the content of this module:



- What is a cache?
- Why are caches used at all?
- What areas of the system might benefit from the use of a cache?
- How do you tune a cache?

Additional Resources



Additional resources – The following references can provide additional details on the topics discussed in this module:

- Wong, Brian L. 1997. *Configuration and Capacity Planning for Solaris Servers*. Palo Alto: Sun Microsystems Press/Prentice Hall.
- *The SPARC Architecture Manual Version 9*. Prentice Hall.
- Catanzaro, Ben. 1994. *Multiprocessor System Architectures*. SunSoft Press/Prentice Hall.



What Is a Cache?

- Keeps accessed data near its user
- Must provide high-speed data access
- Holds a small subset of the available data
- Is used by hardware and software
- Has many different types around the system
- Is critical for performance
- Can be managed in many different ways

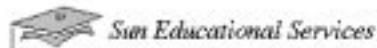
What Is a Cache?

Caches are used to keep a (usually small) subset of active data near where it is being used. Just as the reference books you use tend to remain on your desk, the system tries to keep data being used near the point of use. This provides the ability to not only access the data more quickly, but avoids a time-consuming retrieval of the data from its current location.

Caches are used by the hardware, in the CPU and I/O subsystem, and the software, especially for file system information. The hardware and software contain dozens of caches.

A cache usually can hold only a small subset of the data available, sometimes less than .01 percent, so it is critical that a subset contain as much of the data in use as possible.

There are many different ways to manage a cache, as you will see, but there are only a few that are commonly used.



Hardware Caches

- There are hardware caches all over the system
 - CPU level one and level two
 - PDC/TLB/DLAT
 - Main memory
 - I/O components
- Each system has different caches with different characteristics

Hardware Caches

The most commonly known hardware caches are the CPU caches. Modern CPUs contain at least two data caches: level one and level two.

The level one cache is also called the on-chip or internal cache. It is usually on the order of 32 Kbytes in size. The level two cache is also called the external cache. It is the most commonly known cache.

In addition, main storage can be considered a cache, as it holds data from disks and the network that the CPU might need to access.

CPUs usually also contain a cache of virtual address translations, called the Translation Lookaside Buffer (TLB), Dynamic Lookaside Table (DLAT), or Page Descriptor Cache (PDC), which will be discussed later.

In addition, the hardware I/O subsystem has various caches that hold data moving between the system and the various peripheral devices.



SRAM and DRAM

- Hardware caches are usually SRAM.
- Static RAM does not need a refresh.
 - DRAM data fades after about 10 cycles.
 - Refresh rewrites data, but this delays access.
 - SRAM does not fade, so no refresh is needed.
- SRAM takes four times as many transistors as DRAM.
 - SRAM will always be more costly and take more power.

SRAM and DRAM

Most caches are made from SRAM (static random access memory). The system's main storage is made from DRAM (dynamic RAM). Static RAM uses four transistors per bit, while DRAM uses only one.

It would seem that using DRAM everywhere would be a good idea; it takes 1/4 the space, so there could be four times the memory. But DRAM has one problem – the electrical charge that gives it its values fades (quickly) over time. Usually every 10 or so system clock cycles the contents of the DRAM must be refreshed. Refresh involves reading the data into the memory controller and writing it back fresh. Since the memory being refreshed is busy during this process, data cannot be read from nor written to it. Given the cost of accessing main memory, this cost is usually hidden.

SRAM does not require a refresh; it refreshes itself. As a result, the data it contains is always available, making it ideal for locations that require high-performance access such as caches.



CPU Caches

- There are usually two levels of cache in a CPU:
 - Level one (internal) – Usually 4–40 Kbytes in size
 - Level two (external) – Usually .5–8 Mbytes in size
- A level one cache operates at CPU speeds.
- Data must be in the level one cache before the CPU can use it.

CPU Caches

As mentioned previously, there are usually two levels of CPU cache, level one and level two.

The level one cache is usually from 4–40 Kbytes in size, and is constructed as part of the CPU chip. Of the three million transistors on the UltraSPARC-II™ CPU chip, two million are used for the cache.

The level two cache is usually from 0.5–8 Mbytes in size, and is located near, but not on, the CPU.

The level one cache operates at CPU speeds since it needs to provide data to the CPU as rapidly as possible. Data must be in the level one cache before the CPU can use it.



Cache Operation

The cache's user requests data.

- If found, this is a cache *hit*. The data is returned.
- If not found, this is a cache *miss*.
 - Data is requested from the next level up.
 - The requester has to wait until the data arrives.

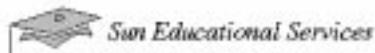
Cache Operation

All caches, hardware and software, operate in a similar fashion. A request is made to the cache controller or manager for a particular piece of data. That data is identified by its *tag*, which could be its address or some other uniquely identifying attribute.

If the data is found in the cache, it is returned to the requester. This is known as a cache *hit*.

If the data cannot be located in the cache, it must be requested from the next level of storage up the chain. This is known as a cache *miss*. The requester must wait, or process other work if it can, until the requested data becomes available. The length of time that the requester must wait depends on the construction of and activity in the system.

The system uses many techniques to improve the hit rate, which are discussed in the following pages.



Cache Miss Processing

- Caches are usually kept 100 percent full.
- On a cache miss, new data must be put in the cache.
 - This process is called a *move in*.
- A location for the incoming data must be chosen.
- Almost always the *least recently used* (LRU) algorithm is used.
 - The oldest data is always replaced.

Cache Miss Processing

For efficiency reasons, caches are usually kept 100 percent full. This way, as much data as possible is available for high-speed access.

When a cache miss occurs, new data must be put in the cache. This process, called a *move in*, must replace data already in the cache.

This replacement process requires that a location for the incoming data be found, at the expense of some data already present in the cache.

Depending on the design of the cache, there may be only one place that the new data can go, or there may be many. If there is more than one, a location least likely to hurt performance is chosen.

The algorithm used to find the new location almost always assumes that the oldest unreferenced candidate location (the longest unused) is the least valuable and will choose it as the location. This type of algorithm is known as LRU (least recently used).



Cache Replacement

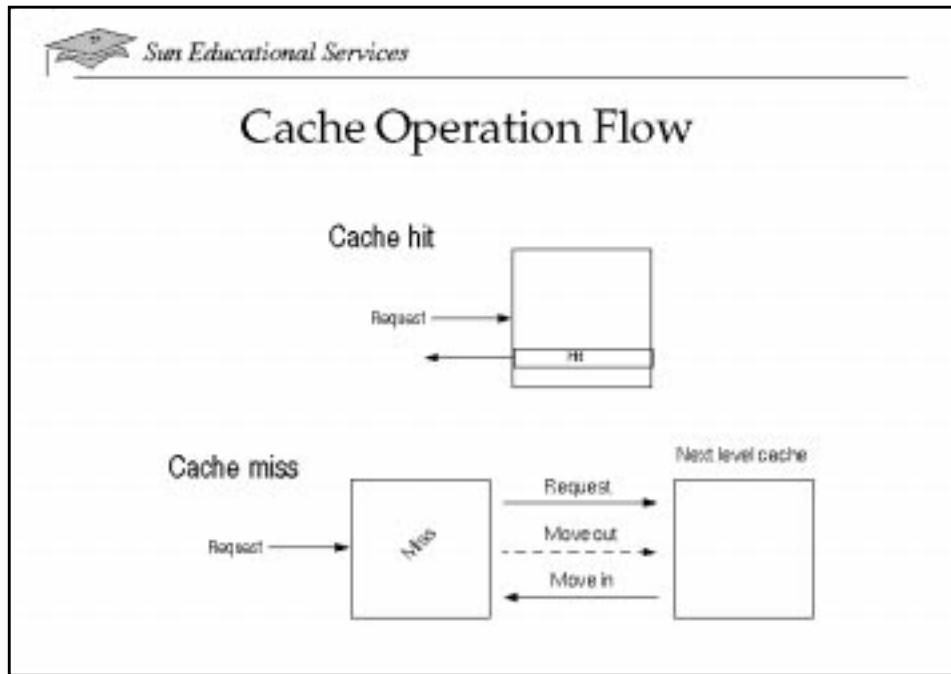
- Data to be replaced may have to be copied back.
 - Changes are not always reflected up.
 - A *move out* or *flush* copies modified data to the next level up.
- Once the move out is complete, the move in starts.
- The amount of data moved is a *cache line*.
 - A hardware cache line is usually 64 bytes.

Cache Replacement

When a location for the incoming data is found, the system cannot just copy the new data into this location; the incoming data could overlay the only current copy of the existing data. In this case, the data must be *moved out*, or *flushed*, and copied to the next level up. If the next level up already has a current copy of the data, it can be *abandoned*, and the new data copied directly in.

Depending on the hardware design, the move in can begin before the move out has finished.

The amount of data that is moved is called a *cache line*. A cache line, in the CPU caches, is usually 64 bytes, but sometimes only 16. In other types of caches, it can be as small as 1 byte or as large as several Kbytes or more.



Cache Operation Flow

In summary, a cache works as follows.

Data is requested from a cache.

1. If the data is present, the cache line (or portion of the line) is returned to the requester.
2. If the data is not present, the requestor must wait until the data has been received from the next level up. When the data is available, a cache location is found for it, and the existing data in that location is either abandoned or moved out, depending on its state at the next level up. A full cache line of data will be transferred.
3. If the next level cache cannot find the data, it requests the data from its next level up, until the data is found and moved in or an error is generated.



Cache Hit Rate

- The performance of a cache depends on its hit rate.
- The cache hit rate is how often requested data is available.
- The cache hit rate depends on:
 - Size of the cache
 - Fetch rate
 - Locality of data references
 - Cache structure

Cache Hit Rate

The cache hit rate is the percentage of memory accesses that result in cache hits or finding the requested data in cache.

The cache hit rate depends on:

- The size of the cache – How much room there is
- The cache's fetch rate – How much data is placed in the cache for each memory access
- Locality of reference – How close newly referenced data is to currently used data
- Cache structure – How the cache is built and managed



Cache Hit Rate

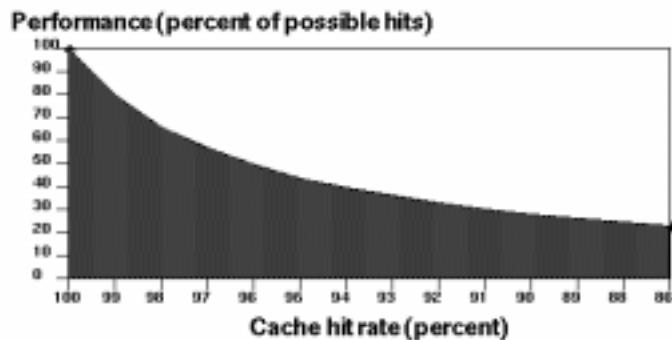
- Misses are very expensive.
- For example:
 - Hit cost is 20 units, miss cost is 600 units.
 - At 100 percent hit rate, cost = $100 \times 20 = 2000$.
 - A 99 percent hit cost = $(99 \times 20) + (1 \times 600) = 1980 + 600 = 2580$.
 - A one percent miss cost is $580 + 2000$ or 29 percent degradation.
- The exact numbers depend on the system.

Cache misses can be very expensive, due to the much higher cost of getting data from the next level up.

Even a miss rate of just a few percent can make a noticeable difference in a system's performance.



Effects of CPU Cache Misses



There are many ways to improve the hit rate.

Effects of CPU Cache Misses

Performance degrades very quickly as the hardware cache miss rate increases.

There are many ways to improve the hit rate, most involve the design characteristics of the cache. Different characteristics can be used depending on system requirements and cost considerations.



Cache Characteristics

There are several pairs of mutually exclusive cache characteristics.

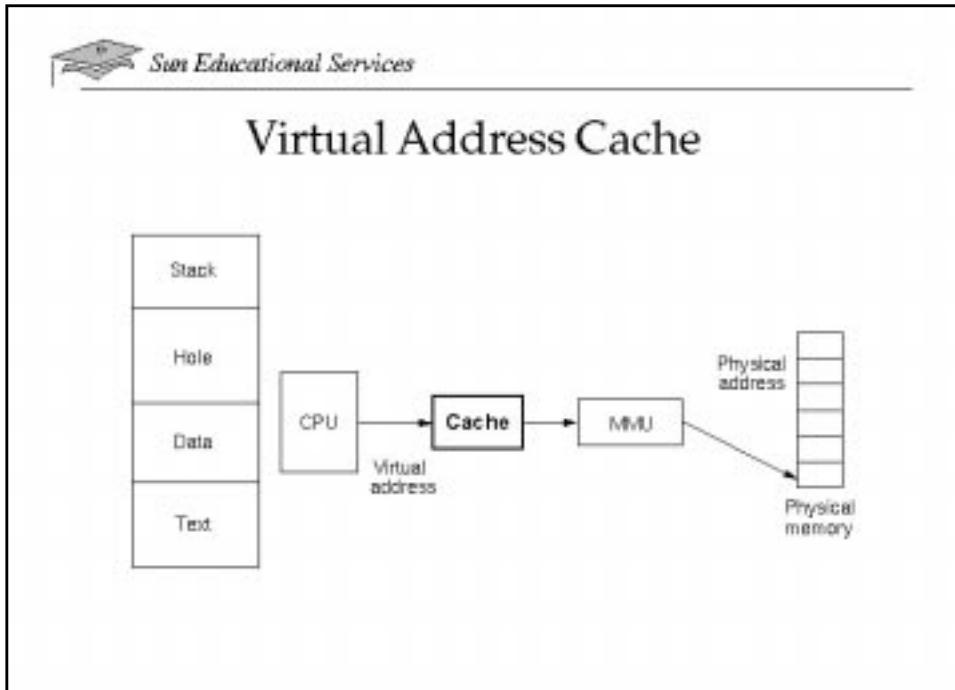
Virtual	Physical
Write through	Write back
Direct mapped	Set associative
Fully associative	
Harvard	Unified
Snooping cache	

Cache Characteristics

There are several major cache design characteristics that can be chosen, depending on the system's requirements.

Most of these characteristics apply to software caches as well, although they might be implemented differently than they are in hardware caches.

Once you identify a cache and its characteristics, you can determine how to manage it for better performance.



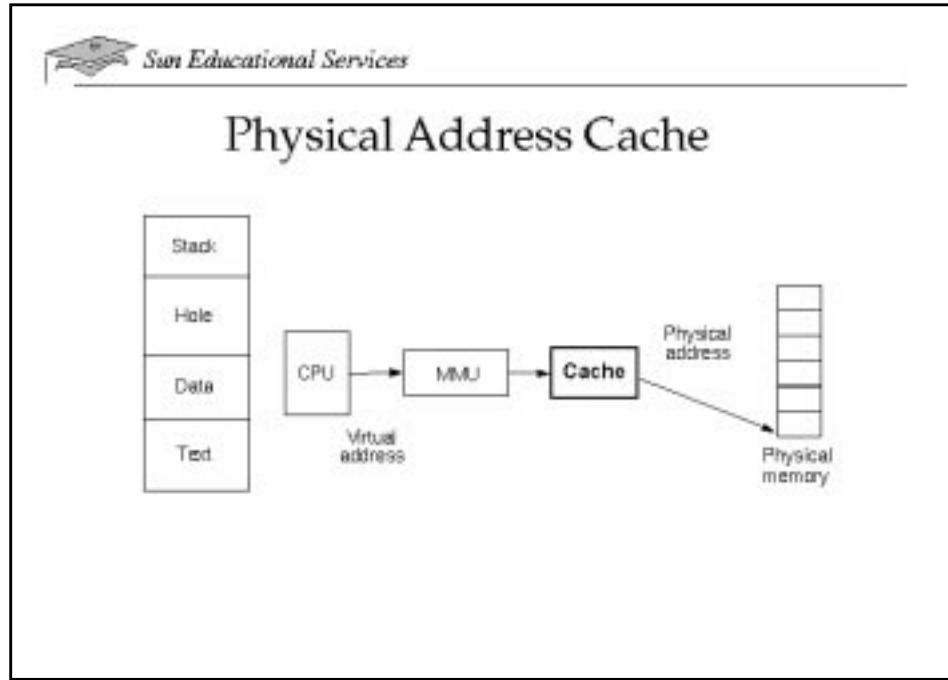
Virtual Address Cache

A virtual address cache can be found in some early SPARC desktop designs (the sun-4c architecture systems) and the UltraSPARC™ systems.

A virtually addressed cache uses virtual addresses to determine which location in the cache has the requested data in it.

The advantage to this scheme is that the memory management unit (MMU) translation can be avoided if the data is found in cache. This means that the address translation mechanism can be bypassed.

The disadvantage is that cache aliasing problems can arise. These problems cause such a performance impact that the virtual address cache is usually impractical. By careful management of *MMU contexts* these problems can be avoided.



Physical Address Cache

A physically addressed cache is one that uses the physical address of a piece of data to determine where in cache it could be located.

A disadvantage to this type of cache is that for every address accessed by the CPU, an MMU translation must be located, regardless of whether the data is in cache. This can be quite expensive, especially in a very busy CPU, but these translations are also cached.

However, a physical address cache pretty much eliminates cache aliasing problems, the drawback with virtual address caches, at the expense of MMU references.



Direct Mapped and Set Associative Caches

- *Direct-mapped* cache – One possible location for the data
- *Set-associative* cache – Multiple possible locations
 - All possible locations are checked in parallel.
 - The number of locations is referred to as *ways*; for example, four-way.

Set associative caches are more complex, but provide better contention management.

- *Harvard* cache – Separate instructions and data

Direct Mapped and Set Associative Caches

In a *direct mapped* cache, each higher-level memory location maps directly to only one cache location.

In a *set associative* cache, a line can exist in the cache in several possible locations. If there are n possible locations, the cache is known as n -way set associative. While set-associative caches are more complex than direct-mapped caches, they allow for a more selective choice of data to be replaced.

A set-associative cache reduces contention for cache resources. When data must be stored in the cache, it can be stored in one of several locations, which allows other addresses that map to the same cache lines to remain in cache. In a direct-mapped cache, since an address can only map to a single cache line, there is no way for two addresses to share the line without flushing the cache.

Direct-mapped caches are normally used for level two or external caches. Since the logic used for set-associative cache is much more complex than direct-mapped caches, it is normally used for level one or on-chip caches where the size of the cache is more limited but the complexity of the cache logic can be enhanced more easily. The extra performance benefit is more noticeable in this type of cache.

Harvard Caches

A Harvard cache (which is named after the university where it was invented) separates a cache into two parts. The first part, named the I-cache (IS), contains only instructions. The second part, named the D-cache (DS), contains only data. The two parts are usually four-way set associative, and do not have to be the same size (or type).

The idea behind this divided cache is that instructions will not lose their cache residence when large amounts of data are passed through the cache. This allows better performance for the system as a whole, although there may be slight performance degradation for a few applications. Harvard caches are usually used in CPU level one caches.

Caches that do not divide the cache are called unified. Following these naming conventions, the unified (non-Harvard) external cache is named the ES.



Write-Through and Write-Back Caches

- *Write-through* cache – Always updates the next level up when its data changes
- *Write-back* cache – Usually updates only when the line is flushed
 - Write-back caches assume a line modified once will be modified again.
 - This may allow *write cancellation*.
 - Unnecessary updates are avoided, saving bus capacity.

Write-Through and Write-Back Caches

A *write-through* cache always writes a cache line to the next level up any time its data is modified.

A *write-back* cache writes only to cache. It writes to the next level only when the line is flushed or another system component needs the data.

The advantage to a write-through cache is that it ensures that the cache contents are consistent with that of main memory. The disadvantage is there is a lot of extra memory traffic due to the loss of *write cancellation*.

Write Cancellation

A write-back cache takes advantage of a system behavior called *write cancellation*.

The system assumes that, if a particular cache line or data block has been written to once, it can be written to again. If the intermediate writes can be avoided, I/O or memory traffic will be reduced and ultimately the system will not be doing useless work moving out data that will be replaced before it is used again. This is the advantage of a write-back cache.

Write cancellation occurs when the system holds modified data, without writing it up to the next level, potentially cancelling unnecessary writes.



CPU Cache Snooping

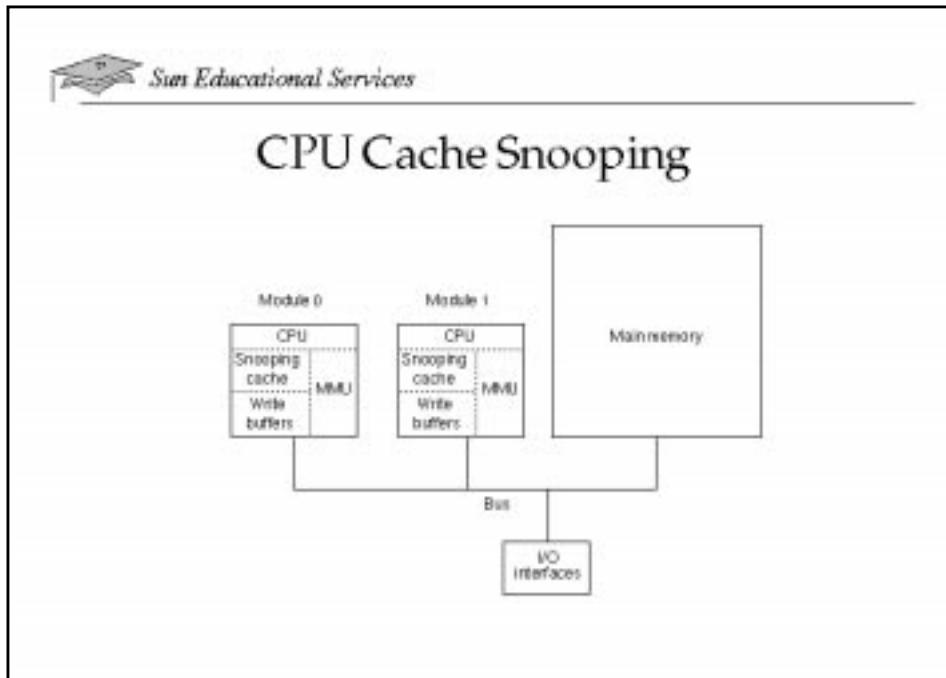
- Write-back caches are great for CPU performance.
- In a multiprocessor system, other CPUs might need modified data from a different CPU's cache.
- To avoid write-through caching, you must have the system use cache snooping.
- All memory requests are monitored by the cache controllers.
 - When a request is made for modified data in its cache, the controller provides it.

CPU Cache Snooping

Write-back caches create an interesting problem in a multi-processor system: cache coherency must be maintained across all of the system's CPUs. This means that each CPU must be able to obtain current, modified data, even if the lower-level cache has not written it to main storage where it is directly available to the other parts of the system.

The system uses a mechanism called *cache snooping* to avoid getting incorrect data. All system write-back caches are responsible for making sure that any component that requests data they have modified gets the current data and not obsolete data from main storage or elsewhere.

The cache controllers do this by monitoring *every* memory request made by every system component. When they observe a request or operation that involves data contained in their cache(s), they respond accordingly.



If a CPU modifies a cache line, it causes all other caches to discard that block if it is present in them.

If the CPU tries to access a line not present in its cache, it needs to read it from a higher level. The CPU requests (via the system bus) the most recent cache line from any cache or from memory.

If the CPU modifies a line, it will cause all other caches to either discard the block (called *write invalidate* protocol) or request all other caches to update their copy of the cache line (*write update* protocol). Which is done is processor architecture dependent.



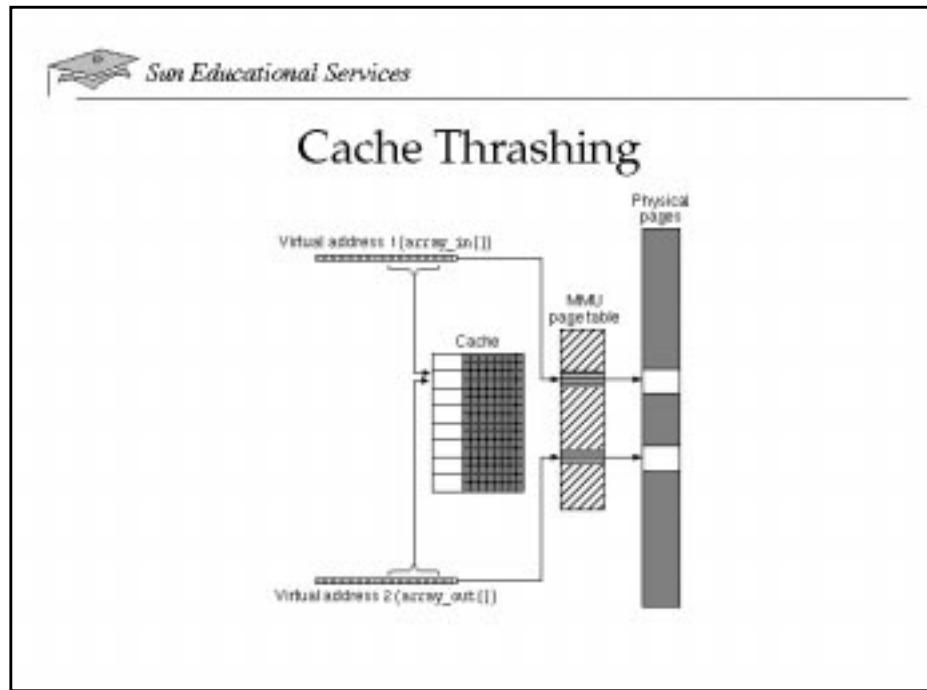
Cache Thrashing

- Picking the wrong memory *stride* essentially turns off the cache.
- Accessing data at a regular address interval can map to the same cache location every time.
 - This can cause 100 percent cache miss.
 - The performance degradation is easily noticeable.
- Diagnosing this problem can be hard; everything *is* working correctly.
- Inspecting the application and comparing it with the cache characteristics is required.

Cache Thrashing

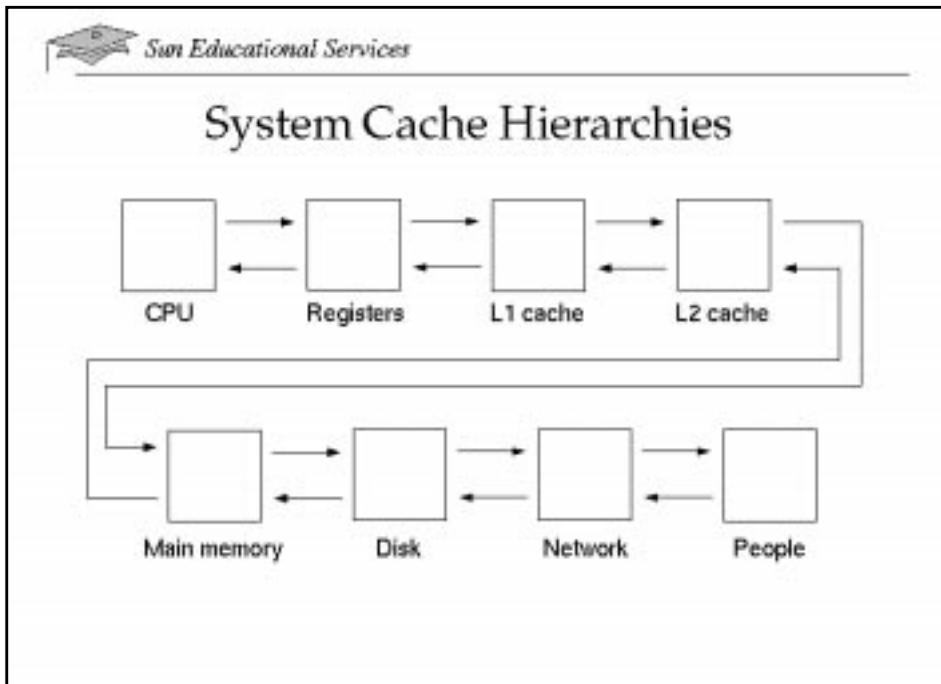
Cache thrashing occurs when the same location or locations in the cache are repeatedly used, causing a move in essentially every time the cache is referenced. When this occurs, the hit rate goes to near zero, with the corresponding immense performance degradation.

This normally occurs when arrays are accessed in memory with a *stride*, or spacing, that corresponds to the cache location mapping algorithm. The stride in this case is almost invariably a power of two.



Diagnosing this problem is usually very simple. By examining the source code for structures that are a power of two in size, the problem can be quickly identified. To solve it, change the structure size.

Also, this problem can occur on one type of system or CPU, but not on another type. If you suspect a problem of this type and the program's source code is not available, move the program to a different type of system to see if the problem disappears.



System Cache Hierarchies

When looked at in a linear fashion, it can be seen that almost every component of the system that contains data is part of a hierarchy of caches.

The further from the CPU the cache is, the larger and slower it is. These components may have different behaviors, but they all operate according to the basic cache principles.



Sun Educational Services

System Cache Hierarchies

Cache	Size	Access Time	Managed by
Registers	1 Kbyte	3 ns	Compiler
L1 CPU	~32 Kbytes	10 ns	Hardware
L2 CPU	.5–8 Mbytes	60 ns	Hardware
Main memory	16 Mbytes to 64 Gbytes	450 ns	Software
Disk	1 Gbyte to 11 Tbytes	20 ms	Software, people
Network	N/A	200 ms	Software, people

The table in the above overhead shows the relative speeds of the different levels of the cache hierarchy. Clearly, data should be at the lowest possible locations when being used; there are significant performance differences between the levels. The further away from the user the data is, the higher the cost of a cache miss.



Relative Access Times

If one nanosecond was one second, then:

Level	Nanoseconds	Seconds
CPU cycle time	3	3
Level 1 cache	10	10
Level 2 cache	60	1 minute
Main memory	450	7.5 minutes
Disk	20,000,000	7.7 months
Network	200,000,000	6.5 years

Relative Access Times

It is difficult for humans to comprehend a nanosecond, one billionth of a second. By converting the nanosecond times from the previous table into "human" times, it is possible to get a better understanding of the cost of a cache miss.

As you might imagine from looking at the relative access times in the above table, the system tries very hard to avoid doing I/O operations. (You will see how this is done in later modules.) Caches figure prominently in these processes.



Cache Performance Issues

- Cache and locality of reference are important.
 - Remember, linked lists are bad for locality.
 - Use SPARCworks™ Analyze to locate problems.
- Cache thrashing can occur.
 - Check algorithms for power of two sizes.
 - Run on a different type of system.
- `vmstat -c` reports cache flushes (but not hits).

Cache Performance Issues

Some performance issues of caches are:

- Locality of reference – Data used together should be stored together to take advantage of cache presence and pre-fetch (bringing data into cache before it is requested). Poor data distribution (bad locality) will cause expensive cache misses.
 - ▼ Linked lists are bad for locality. Use arrays whenever possible.
 - ▼ SPARCworks™ Analyze can show access patterns for a program's routines and data.
- Cache thrashing – If a job runs extremely slowly, and there are no other obvious causes, suspect cache thrashing. Inspect the source code or try the program on a processor with a different CPU architecture.
- Cache flushing statistics – The `-c` option of `vmstat` will give you detailed information on hardware cache flushes. It is of limited use since neither cache hits nor the reason for the flush are reported.

```
# vmstat -c
flush statistics: (totals)
    usr      ctx      rgn      seg      pag      par
    821      960       0       0  123835       97
```

This output contains the following fields:

- usr – User cache type
- ctx – Context cache type
- rgn – Region cache type
- seg – Segment cache type
- pag – Page cache type
- par – Partial-page cache type



Sun Educational Services

Things You Can Tune

- Application
 - Locality of reference
 - Memory strides
 - Algorithms
 - Compiler optimization
- Main memory management

Things You Can Tune

Most of the things that you can tune to affect hardware cache performance can only be done at the application level. Unlike the OS software caches, there are no controls available to change the configuration, size, or behavior of the hardware caches.

You can, however, alter the behavior of main memory, the cache for your disks. This is discussed in the next module.

Check Your Progress

Before continuing on to the next module, check that you are able to accomplish or answer the following:

- Describe the purpose of a cache
- Describe the basic operation of a cache
- List the various types of caches implemented in Sun systems
- Identify the important performance issues for a cache
- Identify the tuning points and parameters for a cache
- Understand the memory system's hierarchy of caches and their relative performances

Think Beyond

Consider the following:

- How could you prove that you were having hardware caching problems?
- Are there workloads that are insensitive to caches?
- How big is too big for a cache?
- How many caches might be too many to coordinate among CPUs?
Why?
- Can you find software equivalents for each of the hardware cache types?

Objectives

Upon completion of this lab, you should be able to:

- Calculate the cost of a cache miss, in both time and percentage of possible performance

Tasks

Cache Miss Costs

Use the following figures to answer the questions in this exercise:

- Cache hit cost – 12 nanoseconds

- Cache miss cost – 540 nanoseconds

1. What speed will 100 accesses run at if all are found in the cache (100 percent hit rate)?

2. What is the cost in time of a 1 percent miss rate? In percentage performance?

3. If a job was running three times longer than it should, and this was due completely to cache miss cost, what percentage hit rate does it achieve?

4. How could the routine in the program that was causing the problem be located?

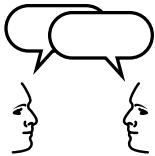
5. What advice would you give the programmer?

Objectives

Upon completion of this module, you should be able to:

- Describe the structure of virtual memory
- Describe how the contents of main memory are managed by the page daemon
- Understand why the swapper runs
- Interpret the measurement information from these components
- Understand the tuning mechanisms for these components

Relevance



Discussion – The following questions are relevant to understanding the content of this module:

- Why do real and virtual memory need to be managed at all?
- What might you need to manage?
- How could you perform this management?
- What other parts of the system depend on the memory system working correctly?

Additional Resources



Additional resources – The following references can provide additional details on the topics discussed in this module:

- Cockcroft, Adrian, and Richard Pettit. 1998. *Sun Performance and Tuning*. Palo Alto: Sun Microsystems Press/Prentice Hall.
- Wong, Brian L. 1997. *Configuration and Capacity Planning for Solaris Servers*. Palo Alto: Sun Microsystems Press/Prentice Hall.
- Man pages for `sar` and `vmstat`.
- Mauro/McDougall. 1999. *Solaris Architecture and Internals*. Sun Microsystems Press/Prentice Hall.



Virtual Memory

- Is what the programs see as addressable memory
- Resides in main memory or on disk (or both)
- Is created and deleted by the program as necessary
- Is managed by the OS
 - SunOS acts as the main memory cache manager.
 - Is contained in address spaces (processes)
 - Is managed in *segments* of related, contiguous bytes

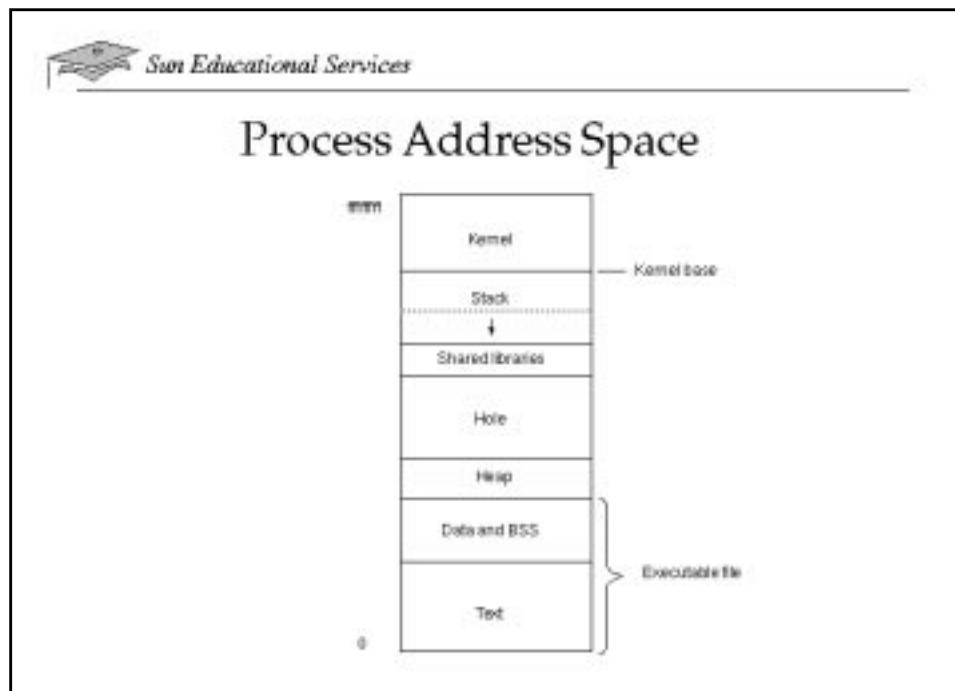
Virtual Memory

Virtual memory is a technique used to provide the appearance of more memory to a program than the system really has. It uses physical main memory as a cache for the total amount of memory in use. The memory that is visible to the program, the amount it has available to use, is the virtual memory.

Virtual memory is created by the OS for the program as it is needed, and destroyed when it is freed (or the process using it ends). If there is no room for a certain section of virtual memory in real, physical memory, it is *paged* out.

Each process gets its own expanse of virtual memory to use, up to a maximum of 3.75 Gbytes. This area, called the process' address space, is where the program runs. Sections of this, called shared memory, can be shared with other programs.

Virtual memory is managed in *segments*, which are variously sized stretches of virtual memory with a related use, such as a shared library, the heap, the stack, and so on.



Process Address Space

The process address space is divided into segments. In a normal address space there are usually at least 15 different segments. Its total size is 4 Gbytes in the Solaris 2.6 software release.

The *text* segment is the executable instructions from the executable file, which are generated by the compiler. The *data* segment contains the global variables, constants, and static variables from the program. Its initial form also comes from the executable file. The block starting with symbol (*BSS*) contains the same types of information that the data segment does; the difference is that the BSS is initialized to zeros, so it is not part of the executable file.

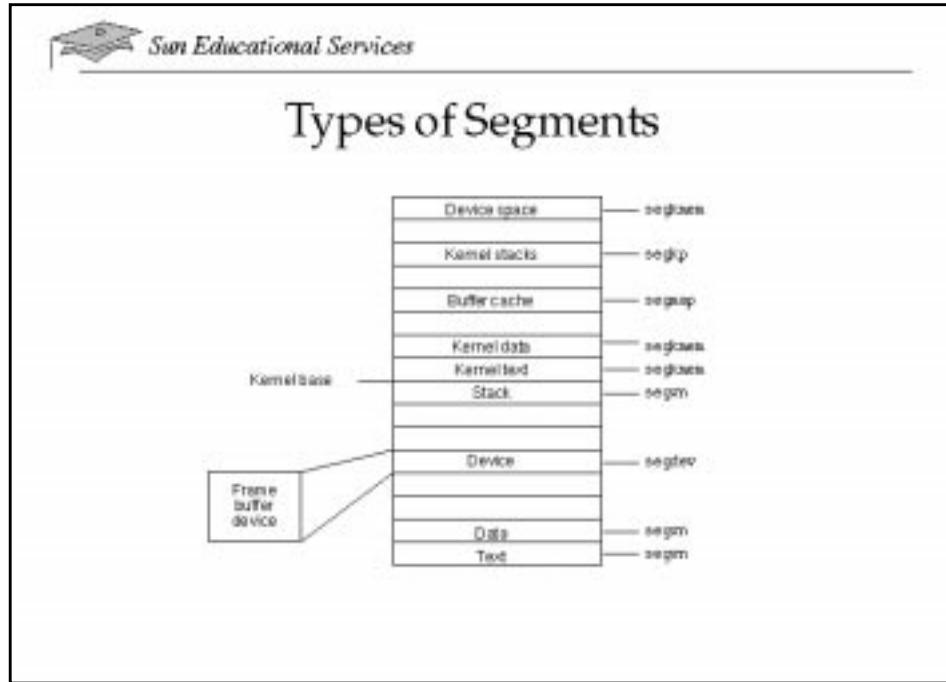
The *heap* is memory that is used by the program for some types of working storage; it is allocated with the `malloc` call.

Following the heap is the *hole*. The hole is that section of the address space that is unallocated and not in use. For most processes, it is the vast majority of the address space. It takes up no resources since it has not been allocated.

Above the heap are the *shared libraries* used by the program. These libraries have the same basic structure as a normal executable file; that is, they contain text, data, and BSS segments. The difference is how the code is compiled. A shared library is compiled as *position independent* so that it can become a section of shared memory that is accessed by every program that wants to use it. Shared libraries, such as `libc.so`, are connected to the program executable as it is loaded into memory when it starts.

Above the shared libraries, at the end of the address space area that the program can address, is the *stack*. The stack is used by the program for variables and storage while a specific routine is running. The stack grows and contracts in size, depending on which routines have been called and what kind of storage requirements each routine has. The stack is normally 8 Mbytes in size.

Located above the stack is the *kernel*. The kernel can be as large as 0.5 Gbytes of virtual memory. While it is a part of the address space, the kernel cannot be referenced by an application. When a program needs access to kernel data, it must be able to run as root or make a system call request. The actual size of the kernel changes, depending on many factors such as peripheral device configuration and amount of system real memory. Essentially the entire kernel is locked in memory: the amount of real memory used is the same as the amount of virtual memory allocated in the kernel.

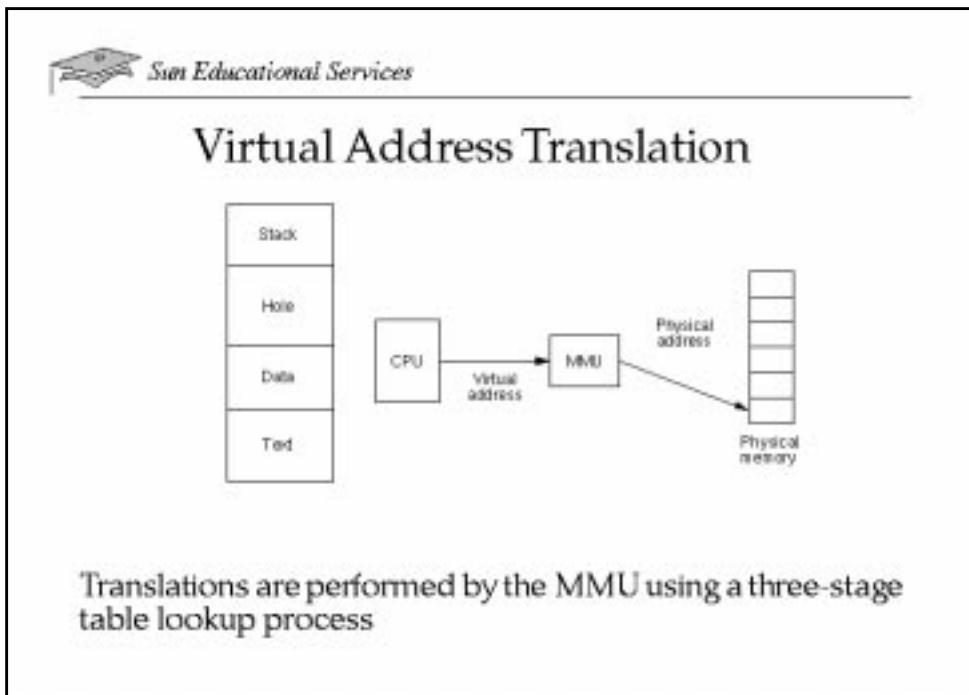


Types of Segments

Segments are managed by segment drivers, just as devices are managed by device drivers. The segment driver is responsible for creating, copying (for `fork`), and destroying the segment, as well as handling page faults for the segment.

There are several different segment drivers; the majority are used to manage kernel memory. The most common segment driver for user processes is the `segvn` driver. The `segvn` driver maps files (represented by vnodes) into the address space. Almost every segment in the address space is backed by a file (an executable, swap, or data file), and the `segvn` driver supports them all.

The usual exception is for a device segment (`segdev`), which represents the frame buffer.

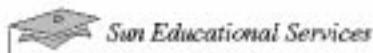


Virtual Address Translation

Since memory acts like a fully associative cache for a disk, a directory mechanism must allow the system to determine whether a page is in memory, and if so, where it is in real memory. This process is called *address translation*.

The address translation is performed (in the recent SPARC systems) as a three-stage table lookup. Parts of the virtual address to be translated are used to index a series of tables, which eventually provide the real address corresponding to the virtual address.

The translation is performed in the CPU's memory management unit (MMU) by the hardware. If any part of the translation cannot be performed, an interrupt occurs, indicating a page fault, which the OS must then resolve. If the address being translated is legitimate, the OS must locate the page containing the requested data.



Page Descriptor Cache

- Is also called TLB or DLAT
- Caches virtual to real address translations
- Holds entries created by MMU table lookups
- Is usually between the CPU and the level one cache
- Has a limited size; usually 64 pages
- Is fully associative
- Has one per CPU
 - Entries are based on the CPU's activity.

Page Descriptor Cache

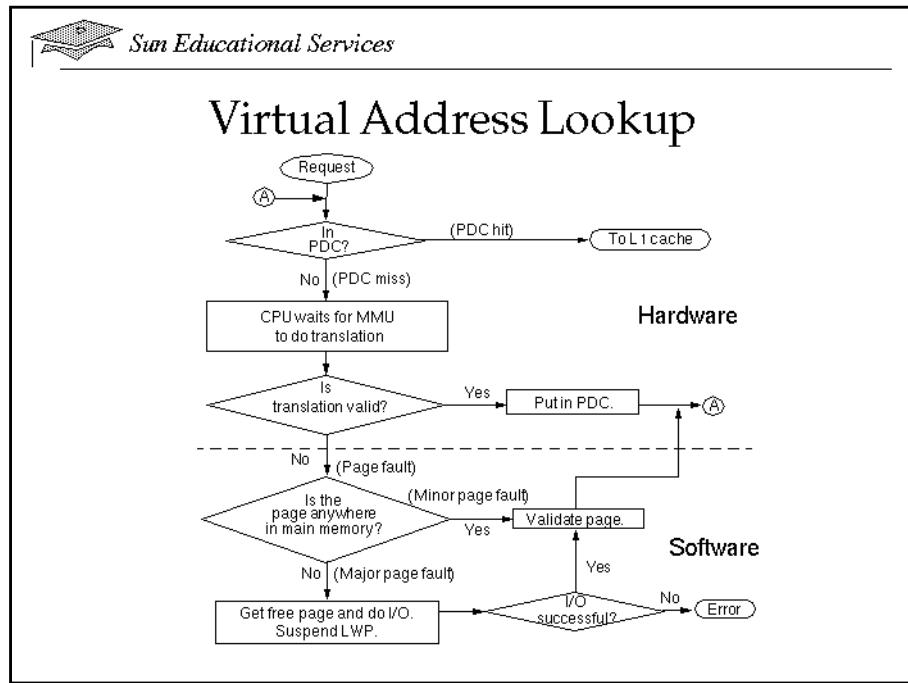
The address translation process is very expensive: it requires three memory accesses before the data can be requested, and must be performed for every instruction or operand reference. If resolving the address of every instruction and operand requires three memory references, or even three cache references, system performance would be exceptionally slow.

The system avoids this by using a cache. Translated addresses are saved in a special cache, usually with 64 entries, called the page descriptor cache (PDC). It is also known as the translation lookaside buffer (TLB) or dynamic lookaside table (DLAT).

Once an address is translated, it is placed in the cache for future reference. Access to the PDC is so fast, it is actually part of the level one cache access time. As usual, the cache is managed on an LRU basis. It is fully associative.

Each CPU (and I/O interface) has its own PDC, located on the CPU chip. The most recent 64 pages referenced by the CPU are contained in each. In a multi-processor system, each PDC's contents will be different, although there is some overlap for shared pages from the kernel and shared libraries.

PDC entries are normally replaced on LRU basis: if a page is not referenced, its entry is eventually replaced by a new page. If the page is referenced again, it is retranslated and reentered into the cache.



Virtual Address Lookup

The diagram above illustrates the process that occurs when a virtual address must be translated.

The PDC is checked first, then a long translation is done. If the page is invalid at the end of the translation, the hardware cannot proceed and asks for help from the software by issuing a page fault interrupt. The OS passes the request to the proper segment driver, which checks memory to see if the page is available. If it is, it attaches the page to the process. (For a shared page, there may already be hundreds of users of the one page.) If the page is not in memory, it requests the appropriate I/O operation to bring in the page (for example, from NFS or UFS).



The Memory Free Queue

- Main memory is a fully associative disk cache managed by the OS.
- Move ins and move outs occur as in any cache.
 - For memory, this is called *paging*.
- Moves involving disk are very expensive.
- A move out followed by a move in is too costly.
- Free memory pages are kept available to avoid the need for move outs before a move in.
- You always need to have "enough" pages on the free queue.

The Memory Free Queue

Main memory is a fully associative cache for the contents of the disks. As such, it has the usual attributes of cache operation, such as move ins and move outs. For main memory, move ins and move outs are called *paging*.

The process for main memory is the same: a cache miss occurs (a page fault). The OS determines that it must read in the page from disk. In the hardware caches, a location is identified, a move out is done, and then a move in (pagein) is done.

This is not done for main memory. Remember, hardware cache operations are quite fast. Pageouts for main memory are disk I/O operations, which are extremely expensive. To avoid the I/O, the virtual memory (VM) system keeps a quantity of free pages which are available to processes needing a new page. This avoids the need to wait for a pageout followed by a pagein; only the pagein is needed.

You must maintain enough free pages for the system's needs.



The Paging Mechanism

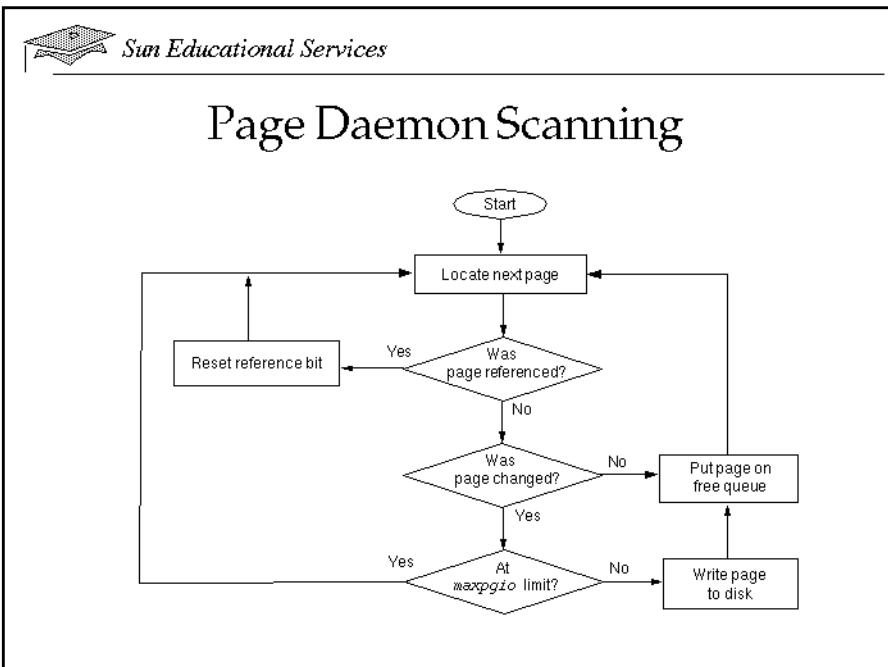
- Every quarter second a check is made to see if the amount of free memory is less than the quantity specified by the *lotsfree* parameter.
- If it is, the page daemon is run to replenish the free memory queue.
- Pages are stolen from their users if they have not been used recently.
- Pages are scanned to determine which can be stolen.
 - The more pages needed, the faster the scan rate.
 - If the page daemon cannot keep up, swapping may occur.

The Paging Mechanism

Pages are not replaced immediately after they are used. Instead, the *page daemon* wakes up every 1/4 second and checks to see if there are enough free pages. If there are, it does nothing. If there are not, it attempts to replenish the free page queue in a process called *page replacement*. The moveouts are asynchronous in this case.

Since main memory is a cache, the algorithm used by the page daemon to replenish the free queue is the usual LRU process. The page daemon inspects, over time, all of the pages in memory and takes those that have not been used.

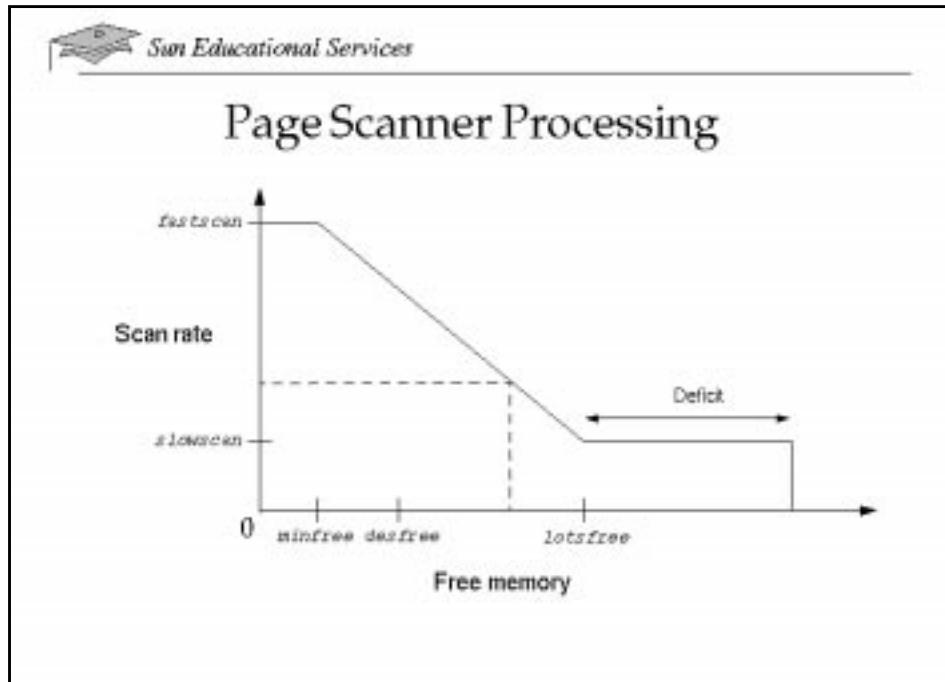
The page daemon LRU process is called *scanning*. The more pages needed, the faster the page daemon scans. The *scan rate*, how many pages are being checked each second, is the best indicator of stress (shortage) in the memory subsystem. If the page daemon cannot keep up with the demand for free pages, the swapper may run, which will temporarily move some memory-heavy work to disk.



Page Daemon Scanning

The page daemon performs the same checks on every page eligible to be taken. There are many types of pages that cannot be taken, such as kernel pages, which are skipped over.

The page daemon checks each eligible page to see if it can be taken. To take the page, it must be unreferenced, and if modified, the *maxpgio* limit cannot have been reached.



Page Scanner Processing

Page daemon processing is conceptually a simple process.

If there is nothing to do, the page daemon does nothing. This determination is made if the number of free pages is greater than *lotsfree*.

If there are less than *lotsfree* pages left, the page daemon begins scanning at the *slowscan* rate. As fewer and fewer pages are left on the free queue, the faster the page daemon scans. When a minimum number (*minfree*) are left, the page daemon scans as fast as possible, at the *fastscan* rate. The scan rate is determined by the number of pages available, as shown in the graph above.

The only special case occurs when too many pages are taken at once, causing a big drop in the free queue. When this occurs, the system temporarily adds to *lotsfree* a quantity (*deficit*) designed to keep this from happening again. The *deficit* value decays to zero over a few seconds.



Default Paging Parameters

Variable	Purpose	Default
<i>lotsfree</i>	Start scanning at <i>slowscan</i> rate	1/64 of main store; minimum 512 Kbytes
<i>desfree</i>	Swapping control	1/2 <i>lotsfree</i>
<i>minfree</i>	Scan at <i>fastscan</i> rate	1/2 <i>desfree</i>
<i>fastscan</i>	Fastest scan rate	1/2 available memory; maximum 64 Mbytes
<i>slowscan</i>	Slowest scanning rate	100
<i>maxpgio</i>	Limit on pages to write per second	60
<i>hand-spreadpages</i>	Alternate tuning mechanism	<i>fastscan</i>

Scanning is capped at a fixed percentage of the CPU capacity.

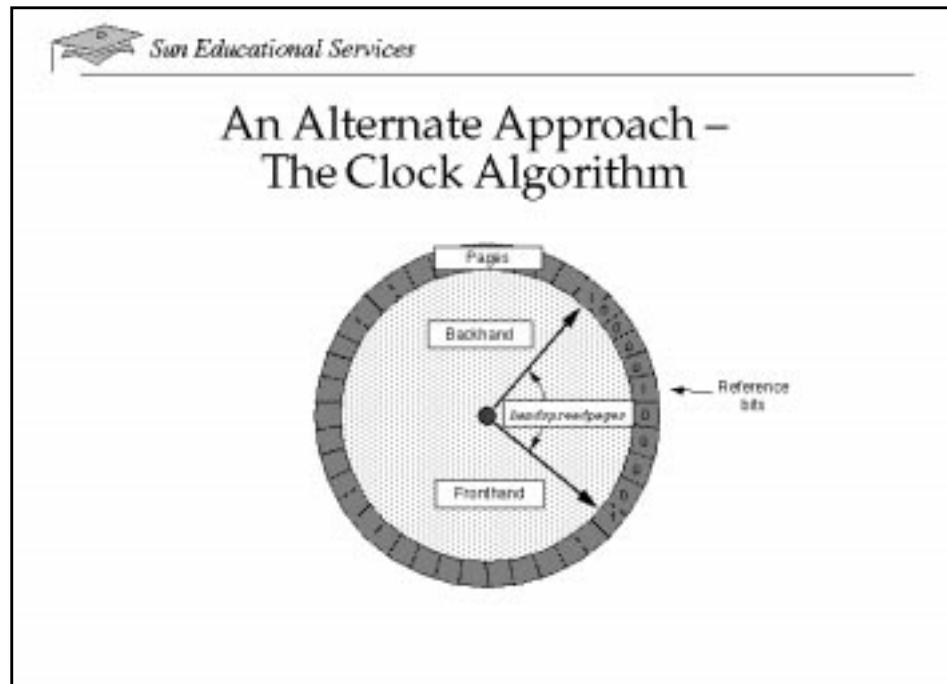
Default Paging Parameters

The table above shows the defaults for the page daemon parameters.

Prior to SunOS 5.6, these parameters were not scaled for multi-processor systems, causing larger systems to default to too small a free queue and unnecessarily high scan rates.

With SunOS 5.6 and higher, these parameters are properly scaled. With *deficit* processing, it is no longer necessary to tune these parameters. If you have values in */etc/system* from an earlier release, remove them when migrating to the Solaris 2.6 environment.

To avoid scanning taking over the CPU on a very memory-stressed system, SunOS 5.6 limits scanning to 4 percent of available CPU power, regardless of the calculated scan rate.



An Alternate Approach – The Clock Algorithm

The page daemon also has an alternate mechanism for controlling paging, using the parameter *handspreadpages*.

The page daemon always uses both the *lotsfree* and *handspreadpages* parameters when it does its work. However, the operation of the page daemon can be tuned through both.

It is always better to choose one or the other method, since it becomes difficult to predict system behavior if both are tuned.

The *handspreadpages* parameter governs how long after a page has been checked for usage it is checked again. By default, each page is checked every 1/2 scan. By using the *lotsfree* and *minfree* parameters to control the scan rate, you can easily determine how long a page will remain unused before it is stolen.

The length of time a page goes before it is checked again is directly related to the scan rate.

You can change this time by adjusting the *handspreadpages* parameter. Adjust it to less than half of the number of main memory pages to have the page daemon recheck a page when it is that distance past it. This means that the page will be rechecked more quickly, making it more likely that it will be unused, and thus can be stolen. This allows a program to keep the pages for a shorter period of time than the scan parameters by themselves allow.

The drawback is that the page daemon could take pages that the program is still using, causing the program to quickly take them back. This causes memory *thrashing*, which uses CPU time and does not replenish the free queue. Performance usually degrades rapidly at this point, especially if the pages are modified and being written out.

Do not try to adjust both *handspreadpages* and the scan parameters: it becomes too difficult to determine how fast pages are being stolen, and it may cause thrashing.

The default value of *handspreadpages* is the value of *fastscan*, with a default maximum of 64 Mbytes (eight thousand one hundred and ninety two 8-Kbyte pages).



maxpgio

- Modified pages need to be written back.
- If every page was modified, there would be periodic I/O spikes.
 - This would not help performance.
- The system cannot take only unmodified pages.
 - Remember, some modified pages must be taken.
 - Determine how many is safe.
- You must adjust `maxpgio` to a realistic value to set the limit.
 - Start with the number of swap disks $\times 60$.

maxpgio

When a page daemon steals pages, some of those pages may be modified (or dirty). Just like any other cache, when a modified cache line is being replaced, it must be written back to the next higher level cache. In the case of main memory, this means written back to disk.

If a large number of dirty pages were taken every quarter second, this would cause significant I/O spikes, as well as large amounts of "non-productive" I/O that are not related to the system's execution.

To limit the level of these spikes, the system provides a parameter named `maxpgio`. `maxpgio` tells the page daemon how many modified pages it may take every second. It is divided by four to obtain the limit for each invocation of the page daemon.

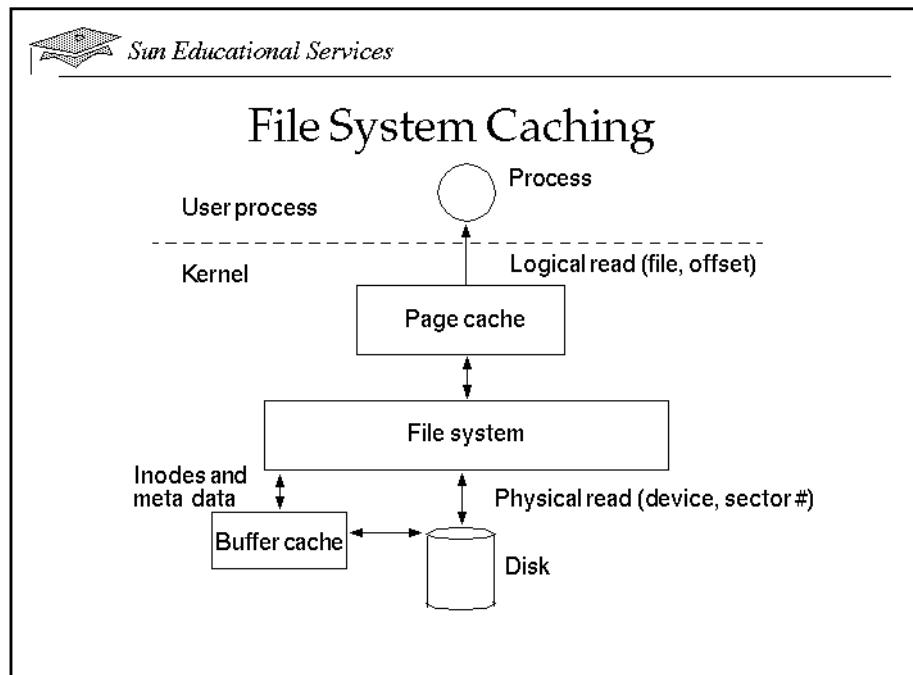
If the page daemon has taken $maxpgio \div 4$ pages, any other pages it takes must be unmodified; if an unused but modified page is found, it is ignored.

maxpgio usually defaults to 60 (40 on desktop systems). This means that only 60 writes of modified pages may be issued per second. (More than 60 pages may be written, however, depending on the number of pages per I/O.) In a small system, this is quite reasonable as 60 I/Os may be a large part of its capacity.

In a larger system, perhaps with hundreds of disks, this default is too small. If the number of modified dirty pages is set too low, the system scan rate will increase as otherwise acceptable modified pages cannot be taken, and the page daemon must continue to look for unmodified pages. The page daemon will not get enough pages, and the scan rate will increase the next time it runs.

To prevent this, you must set *maxpgio* properly. Remember, it covers all stolen and modified pages, whether from a process address space (such as the heap or stack) or from files. If set too high, it can cause I/O spikes or periodic overloads.

The recommended setting for *maxpgio* is $60 \times$ the number of swap disks. This setting may be too low to handle file pages in some cases. You might also try $120 \times$ the number of disk controllers to avoid the focus on swap devices.



File System Caching

File systems, specifically UFSs, are discussed in detail in Module 9, "UFS Tuning." However, the file system cache is not implemented in the file system. In the Solaris environment, the file system cache is implemented in the virtual memory system. Therefore, an explanation of how the Solaris environment file caching works and the interactions between the file system cache and the virtual memory system is relevant to this module.

The Solaris method of caching file system data is known as the page cache. The page cache is dynamically sized and can use all memory that is not being used by applications. The Solaris page cache caches file blocks rather than disk blocks. The key difference is that the page cache is a virtual file cache, rather than a physical block cache. This allows the SolarisOS to retrieve file data by simply looking up the file reference and seek offset, rather than invoking the file system to look up the physical disk block number.

The above overhead shows the Solaris page cache. When a Solaris process reads a file the first time, file data is read from disk though the file system into memory in page-size chunks and returned to the user.

The next time the same segment of file data is read it can be retrieved directly from the page cache, without having to do a logical-to-physical lookup through the file system.

The buffer cache is used only for file system data that is only known by physical block numbers. This data only consists of metadata items – direct/indirect blocks and inodes. All file data is cached through the page cache.

The previous overhead is somewhat simplified. The file system is still involved in the page cache lookup, but the amount of work the file system needs to do is dramatically simplified. The page cache is implemented in the virtual memory system. In fact, the virtual memory system is built around the page cache principle, and each page of the physical memory is identified the same way, by file and offset.

Pages associated with files point to regular files, while pages of memory associated with the private memory space of processes point to the swap device. The important thing to remember is that file cache is just like process memory. File caching shares the same paging dynamics as the rest of the memory system.

The Buffer Cache

- Caches only inodes and metadata
- Is dynamically sized; usually too large
- Is tuned using the *bufhwm* parameter in /etc/system
- Uses a sizing formula for UFS file systems:
300 bytes per inode + 1 Mbyte for every 2 Gbytes of concurrently accessed files
- For its cache hit rate, is monitored using sar -b
- Works best with a read hit rate of 100 percent on file system with few large files and 90 percent on file system with many small files

The Buffer Cache

The buffer cache is used in the Solaris environment to cache inodes and file metadata; it is also dynamically sized. The buffer cache will grow as needed, until it reaches a ceiling specified by the *bufhwm* kernel parameter. By default, the buffer cache is allowed to grow until it uses 2 percent of physical memory. The upper limit for the buffer cache can be displayed by using the sysdef command.

```
# sysdef
*
* Hostid
...
...
* Tunable Parameters
*
1196032      maximum memory allowed in buffer cache (bufhwm)
922          maximum number of processes (v.v_proc)
...
...
60          maximum time sharing user priority (TSMAXUPRI)
SYS         system class name (SYS_NAME)
#
```

Since only inodes and metadata are stored in the buffer cache, it does not need to be very large buffer. In fact, you only need 300 bytes per inode, and about 1 Mbyte per 2 Gbytes of files that are expected to be accessed concurrently.

Note – Buffer cache size of 300 bytes per inode, and 1 Mbyte per 2 Gbytes of files expected to be accessed concurrently is a rule of thumb for the UFS.

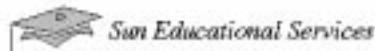
For example, if you have a database system with 100 files totaling 100 Gbytes of storage space and only 50 Mbytes of files are accessed at the same time, then at most you would need 30 Kbytes ($100 \times 300 \text{ bytes} = 30 \text{ Kbytes}$) for the inodes, and about 25 Mbytes ($[50 \div 2] \times 1 \text{ Mbyte} = 25 \text{ Mbytes}$) for the metadata.

On a system with 5 Gbytes of physical memory, the default for *bufhwm* would be 102 Mbytes (2 percent of physical memory), which would be more than sufficient for the buffer cache. You can limit *bufhwm* to 30 Mbytes to save memory by setting the *bufhwm* parameter in the /etc/system file. To set *bufhwm* to 30 Mbytes, add the following line to the /etc/system file (units for *bufhwm* are in Kbytes).

```
set bufhwm=30000
```

You can monitor the buffer cache hit rate using *sar -b*. The statistics for the buffer cache show the number of logical reads and writes into the buffer cache, the number of physical reads and writes out of the buffer cache, and the read/write hit ratios.

Try to obtain a read cache hit ratio of 100 percent on systems with few, but very large files, and a hit ratio of 90 percent or better for systems with many files.



The Page Cache

- Page cache grows to consume all available memory.
- There is no equivalent parameter to *bufhwm* for sizing the page cache.
- Page scan rate is proportional to the rate at which the file system is reading or writing pages to disk.
- Heavy file system usage results in high page scan rates.
- With heavy file system usage, a high page scan rate does not necessarily indicate a memory shortage.

The Page Cache

There is no equivalent parameter to *bufhwm* for the page cache. The page cache simply grows to consume all available memory, which includes all process memory that is not being used by applications.

The rate at which the system pages and the rate at which the page scanner runs is proportional to the rate at which the file system is reading or writing pages to disk. On large systems, this means that you would expect to see large paging values.

Consider a system that is reading 10 Mbytes per second (Mbytes/sec) though the file system; this translates to 1,280 pageins per second. The page scanner must scan enough memory to be able to free 1,280 pages per second. The page scanner must actually scan faster than 1,280 pages per second because not all memory the page scanner comes across will be eligible for freeing. The page scanner only frees memory that has not been recently used. High scan rates are normal if you are using the file system heavily and does not necessarily indicate a shortage of memory.



Priority Paging

- Is included with the Solaris 7 OS but is disabled by default
 - A kernel patch is required for Solaris 2.6 and 2.5.1 OS.
- Is activated by setting *priority_paging* to 1 in /etc/system
- Has a new tuning parameter, *cachefree*, whose default value is twice *lotsfree*
- Starts the page daemon running at *cachefree*
- When enabled, steals only file system pages (data files) until *lotsfree* is reached
- Is very good for large users of random I/O

Priority Paging

Although it may be normal to have high paging and scan rates with heavy file system usage, it is likely that the page scanner will be putting too much pressure on your applications' private process memory.

Pages that have not been used in the last few seconds will be taken by the page scanner when you are using the file system. This can have a very negative effect on application performance. The effect can be poor interactive response, trashing of the swap disk, or low CPU utilization due to heavy paging. This happens because the page cache is allowed to grow to the point where it steals memory pages from important applications. Priority paging addresses this problem.

Priority paging is a new paging algorithm which allows main memory to operate as sort of Harvard cache: providing preferential treatment for instructions (non-file pages) over data (file pages). The priority paging algorithm allows the system to place a boundary around the file cache, so that file system I/O does not cause paging of applications.

Should there be a real memory shortage where there is insufficient memory for the applications and kernel, the scanner is again allowed to steal pages from the applications.

By default, priority paging is disabled. It is likely to be enabled by default in a Solaris release subsequent to the Solaris 7 OS. To use priority paging, you will need either a Solaris 7, Solaris 2.6 with kernel patch 105181-13, or Solaris 2.5.1 with kernel patch 103640-25 or higher environment. To enable priority paging, set the following in /etc/system:

```
set priority_paging=1
```

Priority paging introduces a new additional water mark, *cachefree*. The default value for *cachefree* is twice *lotsfree*. The paging parameters are now:

```
minfree < desfree < lotsfree < cachefree
```

The page daemon now begins stealing pages when free memory reaches *cachefree*, but will only steal file system pages until the amount of free memory reaches *lotsfree*. File system pages do not include those which are shared libraries and executables. If free memory reaches *lotsfree*, the page daemon operates as before, stealing any unused pages (subject to *maxpgio*).

Note – Make sure data files do not have the executable bit set. This can fool the Volume Manager into thinking that these are really executables, and will not engage priority paging on these files.

Priority paging is not appropriate for all systems. It is intended primarily for systems that process large amounts of randomly accessed data files where *madvise* control is ineffective. In these cases, using the normal page daemon algorithm can cause address space pages to be stolen before unneeded file pages, resulting in significant performance problems.

When using priority paging, remember that the page daemon begins processing at *cachefree*, not *lotsfree*, so the free memory queue might be larger than expected under normal conditions.



Available Paging Statistics

Activity	Source cpu_vminfo	vmstat (Unit)	sar (Unit)
Pages scanned	scan	sr (pages)	pgscan/s (pages)
Scanner deficit	deficit	de (pages)	
Pagein requests	pgin		pgin/s
Pages paged in	ppgin	pi (Kbytes)	ppgin/s (pages)
Pageout requests	pgout		pgout/s
Pages paged out	ppgout	po (Kbytes)	ppgouts (pages)
Pages freed	dfree	fr (Kbytes)	pgfree/s (pages)
Page reclaimed from free list	pgfrec	re (pages)	
Average amount of free memory	freetmem	free (Kbytes)	freetmem (pages)
Minor page faults	hat_fault	mf (pages)	
Pages becoming shared			atch/s (pages)

Available Paging Statistics

The table above shows the most commonly used data used in tuning the memory subsystem.

Note the different units reported by the tools. Make sure that you know the memory page size for your system. (Use the pagesize command if necessary.)

```
# sar -g

SunOS rafael 5.7 Generic sun4u      06/11/99

00:00:00 pgout/s ppgout/s pgfree/s pgscan/s %ufs_ipf
01:00:00      0.00      0.00      0.05      0.28      0.00
02:00:00      0.00      0.00      0.00      0.00      0.00
...
...
13:40:00      0.03      0.04      0.06      0.24      0.00
14:00:01      0.42     2.69      5.92     19.69      0.00

Average       0.22      0.95      1.92      6.61      6.06
#
```

The following fields appear in this output:

- pgscan/s – The number of pages, per second, scanned by the page daemon. If this value is high, the page daemon is spending a lot of time checking for free memory. This implies that more memory may be needed.
- pgout/s – The number of pageout requests per second.
- ppgout/s – The actual number of pages that are paged out, per second. (A single pageout request may involve paging out multiple pages.)
- pgfree/s – The number of pages, per second, that are placed on the free list by the page daemon.

```
# sar -p
```

```
SunOS rafael 5.7 Generic sun4u      06/11/99

00:00:00  atch/s  pgin/s ppgin/s  pflt/s  vflt/s slock/s
01:00:00    0.00    0.04    0.05    0.32    0.93    0.00
02:00:00    0.00    0.00    0.00    0.33    0.89    0.00
...
...
14:00:01    0.19    3.05    4.73    2.92   11.22    0.00
14:20:02    0.34    3.68    4.61    0.48   11.44    0.00

Average     0.13    1.09    1.44    0.83    3.85    0.00
#
```

The following fields appear in this output:

- pgin/s – The number of times, per second, that file systems receive pagein requests.
- ppgin/s – The number of pages paged in, per second. A single pagein request, such as a soft-lock request or a large block size, may involve paging in multiple pages.
- atch/s – The number of page faults, per second, that are satisfied by reclaiming a page currently in memory (attaches per second). Instances of this include reclaiming an invalid page from the free list and sharing a page of text currently being used by another process (for example, two or more processes accessing the same program text).

```
# sar -r

SunOS rafael 5.7 Generic sun4u      06/11/99

00:00:00 freemem freeswap
01:00:00      535    397440
02:00:00      565    397538
...
...
14:00:01      195    370126
14:20:02      198    347593

Average       412    385857
#
```

This output includes the following field:

- **freemem** – The average number of memory pages available to user processes over the intervals sampled by the command. Page size is machine dependent.

```
# vmstat 2

procs      memory          page          disk          faults         cpu
r b w    swap   free   re   mf pi po fr de sr f0 s0 s6 --   in   sy   cs us sy id
0 0 0     752  3672   0   3   8   8 15   0   4   0   1   0   0   186  554  200   1   0  98
0 0 0 174640  904   0   2   0   0   8   0   1   0   0   0   0   0   177  469  192   0   0 100
0 0 0 174640  904   0   0   4   0 12   0   2   0   0   0   0   0   178  670  215   0   1  99
0 0 0 174624  904   0   0   0   0   0   0   0   0   0   0   0   0   176  449  187   0   0 100
0 0 0 174624  904   0   0   0   0   0   0   0   0   0   0   0   0   181  467  192   0   0 100
^C#
```

The following fields appear in this output:

- **sr** – The number of pages scanned by page daemon as it looks for pages to steal from processes that are not using them often. This number is the key memory shortage indicator if it stays above 200 pages per second for long periods. This is the most important value reported by **vmstat**.
- **de** – The deficit or anticipated, short-term memory shortfall needed by recently swapped-in processes. If the value is nonzero, then memory was recently consumed quickly, and extra free memory will be reclaimed in anticipation of being needed.
- **pi** – The number of Kbytes paged in per second. A pagein will occur whenever a page is brought back in from the swap device, or into the new buffer cache. A pagein will cause a process to stop execution until the page is read from disk, and will adversely affect the processes' performance.

- `po` – The number of Kbytes paged out per second. A pageout will be counted whenever a page is written and freed. Often this is as a result of the pageout scanner, `fsflush`, or file close.
- `fr` – The number of Kbytes freed per second. Pages freed is the rate at which memory is being put onto the free list by the page scanner daemon. Pages are usually freed by the page scanner daemon but other mechanisms, such as processes exiting, also free pages.
- `re` – The number of pages reclaimed from the free list. The page had been stolen from a process but had not yet been reused by a different process so it can be reclaimed by the process, thus avoiding a full page fault that would need I/O.
- `free` – Size of the free memory list in KBytes. It includes the pages of RAM that are immediately ready to be used whenever a process starts up or needs more memory.
- `mf` – The number of minor faults in pages. If the page is already in memory, then a minor fault simply reestablishes the mapping to it. A minor fault is caused by an address space or hardware address translation fault that can be resolved without performing a pagein. It is fast and uses only a little CPU time. The process does not have to stop and wait as long as a page is available for use from the free list.



Available Paging Data

cpu_vminfo.scan – Number of pages checked by the pageout scanner
cpu_vminfo.deficit – Average amount of memory a new process needs
cpu_vminfo.pgin – Pagein requests
cpu_vminfo.pgpgin – Number of pages paged in
cpu_vminfo.pgout – Pageout requests
cpu_vminfo.pgpout – Number of pages paged out
cpu_vminfo.dfree – The page daemon has freed a page
cpu_vminfo.pgrec – Pages reclaimed from the freelist
vminfo.freemem – Amount of memory on the free queue
cpu_vminfo.as_fault – Address translation faults (page faults)
cpu_vminfo.hat_fault – Number of page faults resolved (pages already valid)
cpu_vminfo.pgfrec – A requested page was found on the freelist

Available Paging Data

The list above shows the origins of the data elements reported in the memory tuning reports.



Additional Paging Statistics

- The Solaris 7 OS provides an extended set of paging counters.
- Counters can be displayed using `memstat` command.
- `memstat` can be downloaded from `ftp://playground.sun.com/pub/rmc/memstat`
- The `memstat` command shows the three types of paging broken out as executable, application, and file.
- `memstat` is used to verify that by enabling priority paging, the scanner frees file pages before freeing executable and anonymous pages.

Additional Paging Statistics

The Solaris 7OS provides an extended set of paging counters that allow you to see what type of paging is occurring. You can now see the difference between paging caused by an application memory shortage and paging though the file system.

The new paging counters are visible with the `memstat` command which can be downloaded from

`ftp://playground.sun.com/pub/rmc/memstat`. The output from the `memstat` command is similar to that of `vmstat`, but with extra fields to break down the different types of paging. In addition to the regular paging counters (`sr,po,pi,fr`), the `memstat` command shows the three types of paging broken out as executable, application, and file. Sample output follows as well as field descriptions.

```
# memstat 3
memory ----- paging ----- -executable- -anonymous- --filesys - - - cpu ---
free re mf pi po fr de sr epi epo epf api apo apf fpi fpo fpf us sy wt id
2080 1 0 749 512 821 0 264 0 0 269 0 512 549 749 0 2 1 7 92 0
1912 0 0 762 384 709 0 237 0 0 290 0 384 418 762 0 0 1 4 94 0
1768 0 0 738 426 610 0 1235 0 0 133 0 426 434 738 0 42 4 14 82 0
^C#
```

The fields in this output are:

- `pi` – Total pageins per second
- `po` – Total pageouts per second
- `fr` – Total pagefrees per second
- `sr` – Page scan rate in pages per second
- `epi` – Executable pageins per second
- `epf` – Executable pages freed per second
- `api` – Application (anonymous) pageins per second from the swap device
- `apo` – Application (anonymous) pageouts per second to the swap device
- `apf` – Application pages freed per second
- `fpi` – File pageins per second
- `fpo` – File pageouts per second
- `fpf` – File pagefrees per second

Observe the difference with priority paging enabled.

```
# memstat 3
memory ----- paging ----- -executable - anonymous - - filesys -- -cpu ---
free re mf pi po fr de sr epi epo epf api apo apf fpi fpo fpf us sy wt id
3616 6 0 760 0 752 0 673 0 0 0 0 0 760 0 752 2 3 95 0
3328 2 198 816 0 925 0 1265 0 0 0 0 0 816 0 925 2 10 88 0
3656 4 195 765 0 792 0 263 0 0 0 2 0 0 762 0 792 7 11 83 0
^C#
```

With priority paging enabled, the scanner is only freeing file pages. You can clearly see from the rows of zeros in the executable and anonymous memory columns that the scanner chooses file pages first.



Swapping

- Swapping is a last resort (*desperation swapping*).
- If the page daemon consistently cannot keep up with the demand for memory, memory use must be cut.
- The number of swaps is not important; the fact that there are swaps is.
- A process will be swapped when its last LWP is swapped out.
 - Its memory will not be freed until then.
- Do not try to tune swaps, try to eliminate them.

Swapping

If the page daemon cannot keep up with the demand for memory, then the demand must be reduced. This is done by swapping out memory users: writing them to disk. Obviously this does not get the system's work done, and so it is only done when there is no other choice. This is usually called *desperation swapping*.

The system uses an algorithm to choose LWPs for swapping. When the last LWP of a process is swapped, the process itself is removed from memory, and all of its pages are freed. Only pages left unused by the swapped LWP are freed before that.

Do not try to tune swapping; the number of swaps is not important. The fact that the system is swapping at all says that there is a severe load on main memory. Swaps should be eliminated, not reduced.



When Is the Swapper Desperate?

The swapper is considered desperate for memory when all of the following are true:

- There is more than one runnable thread.
- `avefree` and `avefree30` are both less than `desfree`.
- The paging rate is excessive or `avefree` is less than `minfree`.

When Is the Swapper Desperate?

The swapper checks once every second to see if it must act. It will swap out if memory has been low for a sustained period (`avefree30`) and is still low (`avefree`).

`avefree` is the average amount of free memory over the last 5 seconds, and `avefree30` is the average amount of free memory over the last 30 seconds. These are used to ensure that the shortage has not been temporary (`avefree30`), and is still ongoing (`avefree`).

The paging rate is considered excessive if the total paging rate (`pginrate` plus `pgoutrate`) is more than `maxpgio`.



What Is Not Swappable

A thread is not swappable if it is:

- In the system (SYS) or real-time (RT) scheduling class
- A process that has a `fork` in progress
- In execution or stopped by a signal
- Exiting or a zombie
- A system thread
- Blocking a higher priority thread

A priority is calculated for each eligible thread left.

What Is Not Swappable

Only LWPs in the time-sharing and interactive classes are swappable, and not all of those are. The system checks for conditions which make it unsafe (such as a `fork` in progress), unwise (the process is ending), or undesirable (a higher priority thread waiting on a lock is blocked), and exempts those LWPs.

Those that pass this screening are then ranked, and the LWP with the highest ranking is swapped out.



Swapping Priorities

- The higher the calculated priority, the more likely the LWP will be swapped.

- The priority calculation is:

$$epri = swapin_time - rss + (maxpgio + 2) - pri$$

- Where

- *swapin_time* – How long since the thread was last swapped

- *rss* – Amount of memory used by the thread's process

- *pri* – The thread's dispatching priority

Swapping Priorities

The system calculates the effective priority of a LWP to be swapped out (*epri*) using the following calculation:

$$epri = swapin_time - rss \div (maxpgio \div 2) - pri$$

The chosen LWP is then swapped out. The swapping of the LWP itself usually returns little memory. Only when the last LWP of a process is swapped, and the memory used by the entire process is taken, does the amount of memory freed become significant.



Swap In

- Once the pressure is off the memory subsystem, swapped out LWPs will be brought back in, when they are ready to run.
- A swapping priority is calculated again.
- The lowest priority swapped-out LWP will be brought back in.
- The process continues until all LWPs are back.
- Memory pressure may cause new swaps.

Swap In

As soon as an LWP has been swapped out, the system tries to bring it back in. As soon as the pressure is off of the memory subsystem, the system will calculate priorities for the swapped out LWPs, this time bringing in one at a second, as long as memory pressure is off, until all have been brought back in.

Swapped-out LWPs will be swapped in only if they are ready to run, regardless of how much available memory there is. This can lead to LWPs reported on the swap queue with a zero scan rate.

The priority calculation is very similar to the swap-out priority calculation. LWPs are brought back in reverse priority order.

If memory pressure continues after an LWP is brought back in, further LWP swap-outs may occur. Usually these will involve other LWPs, as the LWPs just brought back in will have lower swap priorities due to the presence of the *swapin_time* term in the swap-out priority calculation.



Swap Space

- Virtual memory includes most of main memory.
 - Reservable virtual memory is disk swap space plus physical memory, minus the number of kernel pages and a safety factor.
 - Systems with a large amount of physical memory may not require much disk space for swap.
 - You should allocate a primary swap partition large enough to hold a panic dump.
- Swap space is not allocated until the page is written out.
 - If disk space is not available, the page stays in memory.

Swap Space

The amount of virtual memory available to the system to allocate is the total of real memory (less the kernel) and swap space allocated on disk. The system may provide virtual memory to processes until this limit is reached.

Swap space is provided in two steps: *reservation* and *allocation*. The pages written to swap space are those that have no other place to be written to. This includes the stack, heap, BSS, and modified data segment pages. These are called *anonymous* pages, since they have no file to return to.

Virtual memory, and thus (eventually) swap space, is reserved when a new virtual memory segment is created. Each segment driver requests an appropriate amount of virtual memory to be reserved for it. If space is not available, the system call which requested the operation fails. The usual calls that reserve virtual memory are `fork`, `exec`, and (indirectly) `malloc`, which would usually receive an ENOMEM error.

The amount of reservable swap space (available virtual memory) is the total of all swap disk space plus physical memory, less kernel locked pages and a safety zone (in case of under-reservation). The safety zone is the larger of 3.5 Mbytes or 1/8 of main memory.

A page of swap is not allocated until the page has to be written out. This allows the system to group related pages, limiting the number of writes and perhaps the number of reads necessary to bring the pages back in. If no swap disk space is available for the page, the system leaves the page as is in memory.

Since the system can reserve more swap space than the amount of disk space used for swap, when a page needs to be written out, there may not be any available disk space for the page. If this is case, the page is left in memory.

Systems with large amounts of physical memory may need little, if any, swap space since they can do all of their processing in memory. Remember, always have a primary swap partition large enough to hold a panic dump, even if there is no other need for swap space.

There is no performance difference between swap files and swap partitions, although swap files are easier to manage.



tmpfs

- It allows the creation of a virtual memory ramdisk such as /tmp.
- It uses virtual memory like any other user.
- It uses real memory and swap space.
- If used heavily, /tmp can fill main memory.
- You can limit the use of /tmp.
 - The `size` option in the `vfstab` entry is used.
- You must not move it to a hard drive partition.

tmpfs

Another user of virtual memory is tmpfs. tmpfs allows the creation of virtual memory ramdisks, such as /tmp. The use of a tmpfs file system allows virtual memory to be used in place of real disk space, providing significant speedups for programs that use temporary files, such as compilers and sorting.

tmpfs uses virtual memory as any other user, allocating it when it is written to and freeing it when the files are deleted or truncated. tmpfs does not reserve virtual memory. Virtual memory includes main memory, so the use of /tmp causes the use of equivalent quantities of main memory, at least initially. Overuse can lead to a memory shortage and swapping.

You can limit the amount of virtual memory to be used by specifying the `size` option on the tmpfs line(s) in /etc/vfstab. You can also create multiple tmpfs file systems if you want, but remember, all share the system virtual memory.

Some sites have moved `/tmp` to a physical drive because of concerns about the use of real memory by files in `/tmp` or the possible exhaustion of virtual memory by a job using too much `/tmp` space. Given the vast performance difference between real disk and virtual memory, those programs that use `/tmp` will encounter a significant performance degradation.

It is usually best to allow the page daemon to manage the contents of main memory, so that truly unneeded pages can be written, and use the size option to control usage.

There are no monitors that report on the usage of `tmpfs` space. You need to use the `df -k` command.



Available Swapping Statistics

Activity	Source	vmstat (Unit)	sar (Unit)
Average amount of free swap space	swap_avail	swap (amount of unreserved swap, Kbytes)	
Average amount of free swap space	swap_free		freeswap (amount of unallocated swap sectors)
Process swap outs	swpout	so (Kbytes/sec)	swpout/s
Pages swapped out	pgswapout		bswout/s (sectors)
Average number of swapped out LWPs	swpqe	w	swpq-sz
Percentage of the interval any LWP was swapped out	swpocc		swpocc%
LWP swap ins	swapin	si (Kbytes/sec)	swpin/s
Pages swapped in	pgswapin		bswin/s (sectors)

Available Swapping Statistics

The table above shows the most commonly used data used in monitoring the swap subsystem.

Remember, you do not tune swaps, you eliminate them. The presence of any swaps at all indicates stress on the memory system, since it is so hard to induce swapping.

```
# vmstat -S 2
procs      memory          page          disk          faults         cpu
r b w    swap   free   si   so pi po fr de sr f0 s0 s6 --   in   sy   cs us sy id
0 0 0    1640  3656   0   0   8   8 15   0   5   0   1   0   0   187  556  200   1   1  98
0 0 0   174328 1288   0   0   4   0   0   0   0   0   0   0   0   0   191  470  202   0   0 100
0 0 0   174328 1288   0   0   0   0   0   0   0   0   0   0   0   0   177  447  188   0   0 100
0 0 0   174328 1288   0   0   0   0   0   0   0   0   0   0   0   0   180  447  190   0   0 100
0 0 0   174328 1288   0   0   0   0   0   0   0   0   0   0   0   0   186  447  186   0   0 100
^C#
```

This output contains the following fields:

- swap – Amount of swap space currently available in Kbytes. If this number goes to zero, your system will hang and you will be unable to start more processes.

- so – The number of Kbytes/sec swapped out.
- w – The swap queue length; that is, the swapped out LWPs waiting for processing resources to finish.
- si – The number of Kbytes/sec swapped in.

```
# sar -r
```

```
SunOS rafael 5.7 Generic sun4u      06/11/99

00:00:00 freemem freeswap
01:00:00      535    397440
02:00:00      565    397538
...
...
14:00:01      195    370126
14:20:02      198    347593

Average      412    385857
#
```

This output contains the following field:

- freeswap – The number of 512-byte disk blocks available for page swapping.

```
# sar -w
```

```
SunOS rafael 5.7 Generic sun4u      06/11/99

00:00:00 swpin/s bswin/s swpot/s bswot/s pswch/s
01:00:00      0.00      0.0      0.00      0.0      188
02:00:00      0.00      0.0      0.00      0.0      183
...
...
15:20:01      0.00      0.0      0.00      0.0      240
15:40:00      0.00      0.0      0.00      0.0      282

Average      0.00      0.0      0.00      0.0      202
#
```

This output contains the following fields:

- swpot/s – The average number of processes swapped out of memory per second. If the number is greater than 1, you may need to increase memory.

-
- bswot/s – The number of blocks (512-byte units) transferred for swap-outs per second.
 - swpin/s – The number of LWP transfers into memory per second.
 - bswin/s – The number of blocks (512-byte units) transferred for swap-ins per second.

```
# sar -q
```

```
SunOS rafael 5.7 Generic sun4u      06/11/99

00:00:00 runq-sz %runocc swpq-sz %swpocc
01:00:00      1.5      0
02:00:00      1.0      0
...
...
15:20:01      2.1      1
15:40:00      1.7      2

Average       1.6      1
#
```

This output contains the following fields:

- swpq-sz – The average number of swapped out LWPs.
- %swpocc – The percentage of time LWPs are swapped out.



Available Swapping Data

vminfo.swap_avail – Maximum swap space less reserved swap space
vminfo.swap_free – Unallocated swap space: unreserved plus reserved but not allocated
cpu_vminfo.swapout – Number of processes swapped out
cpu_vminfo.pgswapout – Number of pages written out with LWPs swapped out
sysinfo.swpque – Number of LWPs currently swapped out
sysinfo.swpocc – Percentage of the interval that any LWP was swapped out
cpu_vminfo.swapin – Number of LWPs swapped in
cpu_vminfo.pgswapin – Number of pages read in with LWPs swapped in

Available Swapping Data

The list above shows the origins of the data elements reported in the swapping reports.



Shared Libraries

Shared libraries have a longer startup time when a routine is used; however:

- Once the routine is in memory, the system can avoid subsequent I/Os.
- There is a huge savings in both main memory and disk space.
- Tools such as Analyze from SPARCworks can be used to improve locality of reference.

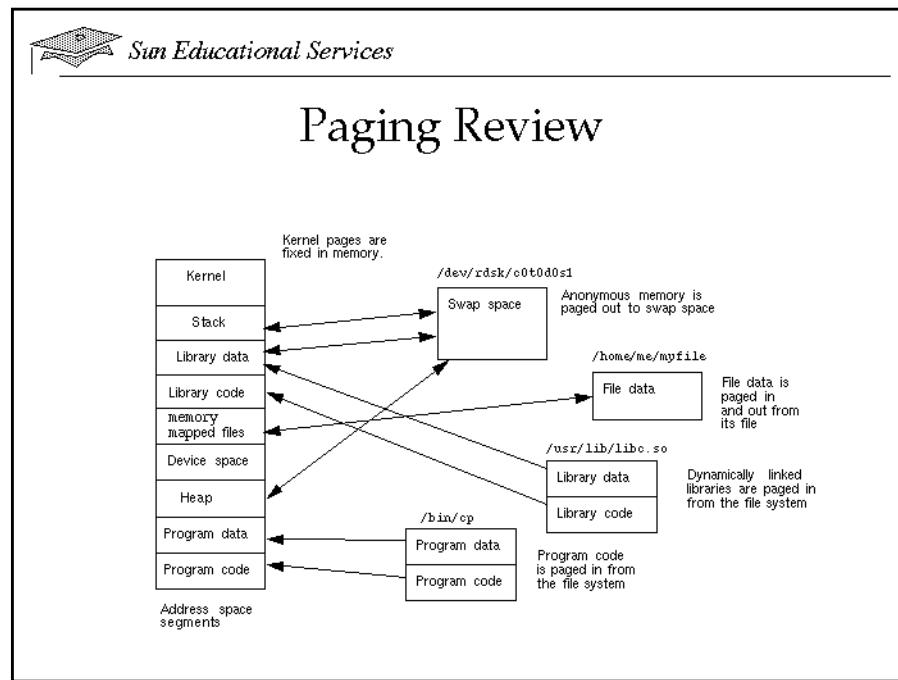
Shared Libraries

Shared libraries can help reduce your memory usage by allowing multiple processes to share the same copy, and thus the same physical memory, of library routines. Indeed, most of the program can be treated as a shared library.

Shared libraries take a small extra amount of time to initialize, but their benefits to the system can be significant. I/O operations to access them on disk are usually eliminated since they remain in memory, physical memory is saved since only one copy is needed, and tuning, such as compiler settings and locality of reference tuning, need to be done only once.

Creation of a shared library requires only the use of special compiler and linker options. No changes to the code are necessary.

There are also maintenance benefits; if a bug is found in the library, it need only be replaced. Programs that use it do not need to be relinked.



Paging Review

The diagram above shows where memory segments that are paged out are written to or originate from. Unmodified pages, such as text segment pages and unmodified data segment pages, are never written out. A new copy is read in from the executable file when needed.



Memory Utilization

- Memory utilization must be observed before application memory requirements can be determined.
- You need to answer the questions:
 - How much physical memory is available?
 - How much memory is being used for file buffering?
 - How much memory is being used for the kernel?
 - How much memory are the applications using?
 - How much memory is free?

Memory Utilization

The amount of memory needed for a system is dependent on the amount of memory required to run your applications. But before an application's memory requirements can be determined, memory utilization must be observed.

Memory in a system is used for file buffering, by the kernel and by applications running on the system. So the first thing to observe in a system is where the memory has been allocated:

- The total amount of physical memory available
- How much memory is being used for file buffering
- How much memory is being used for the kernel
- How much memory my applications are using
- How much memory is free



Sun Educational Services

Total Physical Memory

```
# prtconf
System Configuration: Sun Microsystems sun4u
Memory size: 64 Megabytes
System Peripherals (Software Nodes):

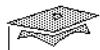
SUNW,Ultra-1
    packages (driver not attached)
        terminal-emulator (driver not attached)
        deblocker (driver not attached)
        cdp-tftp (driver not attached)
        disk-label (driver not attached)
        sun-keyboard (driver not attached)
        ufs-file-system (driver not attached)
...
...
#
```

Total Physical Memory

The amount of total physical memory can be determined from the output of the `prtconf` command.

```
# prtconf
System Configuration: Sun Microsystems sun4u
Memory size: 64 Megabytes
System Peripherals (Software Nodes):
...
...
#
```

As you can see, the physical memory size of this system is 64 Mbytes.



File Buffering Memory

- The page cache uses available free memory to buffer files on the file system.
- A summary of the page cache memory is displayed by the MemTool prtmem command.

```
# prtmem

Total memory:          56 Megabytes
Kernel Memory:         14 Megabytes
Application memory:   19 Megabytes
Executable memory:    17 Megabytes
Buffercache memory:   1 Megabytes
Free memory:          4 Megabytes
```

#

- Page cache is displayed as Buffercache memory - 1 Mbyte.

File Buffering Memory

The page cache uses available free memory to buffer files on the file system. On most systems, the amount of free memory is almost zero as a direct result of this.

To look at the amount of page cache, use the MemTool package. A summary of the page cache memory is available with the MemTool prtmem command.

```
# prtmem
```

```
Total memory:          56 Megabytes
Kernel Memory:         14 Megabytes
Application memory:   19 Megabytes
Executable memory:    17 Megabytes
Buffercache memory:   1 Megabytes
Free memory:          4 Megabytes
```

#

Amount of page cache appears in the output of the prtmem command as Buffercache memory. On this system, the amount is 1 Mbyte.



Kernel Memory

- Use the `sar` command with the `-k` option (kernel memory allocation statistics).
- Total all of the `alloc` columns to determine the amount of kernel memory.
- Remember the output of the `sar -k` command is in bytes.

```
# sar -k 1
SunOS rafael 5.7 Generic sun4u      07/09/99
09:31:33 sml_mem    alloc   fail   lg_mem    alloc   fail   ovsz_alloc   fail
09:31:34 2285568 1848152          0 7815168 6864880          0 1990656          0
#
```

The kernel has grabbed 10,703,688 bytes of memory in this example and is using 10,100,736 bytes.

Kernel Memory

Kernel memory is allocated to hold the initial kernel code at boot time, then grows dynamically as new device drivers and kernel modules are used. The dynamic kernel memory allocator (KMA) grabs memory in large "slabs," and then allocates smaller blocks more efficiently. If there is a severe memory shortage, the kernel unloads unused kernel modules and devices, and frees unused slabs.

The amount of memory allocated to the kernel can be found by using the Solaris `sar -k` command (kernel memory allocation statistics), and totaling all of the `alloc` columns. The output is in bytes.

```
# sar -k 1
SunOS rafael 5.7 Generic sun4u      07/09/99
09:31:33 sml_mem    alloc   fail   lg_mem    alloc   fail   ovsz_alloc   fail
09:31:34 2285568 1848152          0 7815168 6864880          0 1990656          0
#
```

The fields in this output are:

- `sml_mem` – The amount of memory, in bytes, that the KMA has available in the small memory request pool. A small request is less than 256 bytes.
- `alloc` – The amount of memory, in bytes, that the KMA has allocated from its small memory request pool to small memory requests.
- `fail` – The number of requests for small amounts of memory that failed.
- `lg_mem` – The amount of memory, in bytes, that the KMA has available in the large memory request pool. A large request is from 512 bytes to 4 Kbytes.
- `alloc` – The amount of memory, in bytes, that the KMA has allocated from its large memory request pool to large memory requests.
- `fail` – The number of failed requests for large amounts of memory.
- `ovsz_alloc` – The amount of memory allocated for oversized requests (those greater than 4 Kbytes); these requests are satisfied by the page allocator, thus there is no pool.
- `fail` – The number of failed requests for oversized amounts of memory.

Totaling the small, large, and oversize allocations, the kernel has 10,703,688 bytes of memory and is actually using 10,100,736 at present.



Identifying an Application's Memory Requirements

- Predict memory requirements when running more users
- Use pmap command or Memtool's pmem
- Use pmap to show the amount of real memory a process is using
- Use pmap to show the amount of real memory that is shared with other processes on the system, via shared libraries and executables
- Use pmap to show the amount of private (non-shared) memory a process is using

Identifying an Application's Memory Requirements

Knowing how much memory an application uses allows you to predict the memory requirements when running more users.

Use the pmap command or Memtool's pmem functionality to determine how much memory a process is using and how much of that memory is shared.

```
# /usr/proc/bin/pmap -x 10062
10062:  /bin/csh
Address      Kbytes Resident Shared Private Permissions      Mapped File
00010000        144      104      -      104  read/exec      csh
00042000         16       8      -       8  read/write/exec  csh
00046000        248      -      -      -  read/write/exec  [ heap ]
...
...
FF3DC000          8       8       8      -  read/write/exec ld.so.1
FFBE2000        56       8      -       8  read/write/exec  [ stack ]
-----
total Kb      1544     912     760      152
#
```

The output of the `pmap` command shows that the `csh` process is using 912k of real memory. Of this, 760k is shared with other processes on the system, via shared libraries and executables.

The `pmap` command also shows that the `csh` process is using 152k of private (non-shared) memory. Another instance of `csh` will only consume 152k of memory (assuming its private memory requirements are similar).



Free Memory

- Free memory can be measured with the `vmstat` command.
- The first line of output from `vmstat` is an average since boot.
- The real free memory figure is available on the second line. The output is in Kbytes.

```
# vmstat 2
procs      memory          page          disk          faults         cpu
r b w    swap   free   re   mf pi po fr de sr f0 s0 s6 --  in   sy   cs us sy id
0 0 0 210538 2808   0 19 49 38 73   0 24   0 5 0 0 247 1049 239 4 2 94
0 0 0 199664 1904   0 4 12   0 0 0 0 0 2 0 0 224 936 261 7 2 92
0 0 0 199696 1056   7 0 0 628 620 0 20 0 0 0 0 401 1212 412 27 9 64
0 0 0 199736 1144   0 0 0 0 0 0 0 0 0 0 0 165 800 188 0 0 100
^C#
```

Free Memory

Free memory is maintained so that when a new process starts up or an existing process needs to grow, the additional memory is immediately available. Just after boot, or if large processes have just exited, free memory can be quite large. As the file system page cache grows, free memory settles down to a level set by the kernel variable `lotsfree`. When free memory falls below `lotsfree`, memory is scanned to find pages that have not been referenced recently. These pages are moved to the free list.

Free memory can be measured with the `vmstat` command. The first line of output from `vmstat` is an average since boot, so the real free memory figure is available on the second line. The output is in Kbytes.

```
# vmstat 2
procs      memory          page          disk          faults         cpu
r b w    swap   free   re   mf pi po fr de sr f0 s0 s6 --  in   sy   cs us sy id
0 0 0 210528 2808   0 19 49 38 73   0 24   0 5 0 0 247 1049 239 4 2 94
0 0 0 199664 1904   0 4 12   0 0 0 0 0 2 0 0 224 936 261 7 2 92
0 0 0 199696 1056   7 0 0 628 620 0 20 0 0 0 0 401 1212 412 27 9 64
0 0 0 199736 1144   0 0 0 0 0 0 0 0 0 0 0 165 800 188 0 0 100
^C#
```



Identifying a Memory Shortage

- Determine if applications are paging excessively
- Determine if the system could benefit by making more memory available for file buffering
- Use `vmstat` to see if the system is paging
 - If there is no paging, there is no memory shortage.
 - Constant nonzero values in the scan-rate (`sr`) and page-out (`po`) columns are characteristic of excessive paging.
- Look at the swap device for I/Os queued to it
- Use MemTool to measure the distribution of memory

Identifying a Memory Shortage

To determine if there is a memory shortage, it is necessary to:

1. Determine if the applications are paging excessively because of a memory shortage.
2. Determine if the system could benefit by making more memory available for file buffering

Use `vmstat` to see if the system is paging. If the system is not paging, then there is no chance of a memory shortage. Excessive paging is evident by constant nonzero values in the scanrate (`sr`) and pageout (`po`) columns.

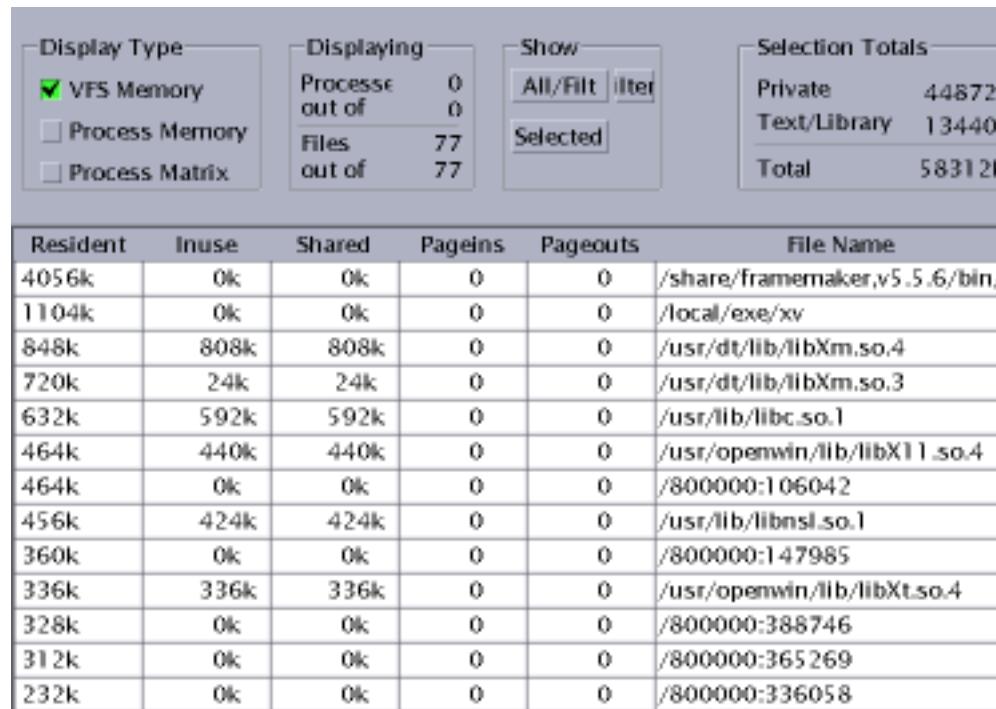
Look at the swap device for activity. If there is application paging, then the swap device will have I/Os queued to it. This is an indicator of a memory shortage.

Use the MemTool to measure the distribution of memory in the system. If there is an application memory shortage, then the file system's page cache size will be very small (less than 10 percent of the total memory available).

Using the memtool GUI

The memtool GUI provides an easy way of invoking most of the functionality of the MemTool kernel interfaces. Invoke the GUI as the root user to see all process and file information.

```
# /opt/RMCmem/bin/memtool&
```



Display Type		Displaying		Show		Selection Totals	
<input checked="" type="checkbox"/> VFS Memory	<input type="checkbox"/> Process Memory	Processes	out of	All/Filt	Iter	Private	44872
<input type="checkbox"/> Process Matrix	<input type="checkbox"/> Process Matrix	Files	out of	Selected		Text/Library	13440
4056k	0k	0k	0	0		/share/framemaker,v5.5.6/bin,	
1104k	0k	0k	0	0		/local/exe/xv	
848k	808k	808k	0	0		/usr/dt/lib/libXm.so.4	
720k	24k	24k	0	0		/usr/dt/lib/libXm.so.3	
632k	592k	592k	0	0		/usr/lib/libc.so.1	
464k	440k	440k	0	0		/usr/openwin/lib/libX11.so.4	
464k	0k	0k	0	0		/800000:106042	
456k	424k	424k	0	0		/usr/lib/libnsl.so.1	
360k	0k	0k	0	0		/800000:147985	
336k	336k	336k	0	0		/usr/openwin/lib/libXt.so.4	
328k	0k	0k	0	0		/800000:388746	
312k	0k	0k	0	0		/800000:365269	
232k	0k	0k	0	0		/800000:336058	

Figure 6-1 MemTool GUI – Buffer Cache Memory

There are three basic modes on the memtool GUI: VFS Memory (Buffer Cache Memory), Process Memory, and a Process/Buffer cache mapping matrix. The initial screen (Figure 6-1), shows the contents of the Buffer Cache memory. Fields are described as follows:

- Resident – The amount of physical memory that this file has associated with it.
- Inuse – The amount of physical memory that this file has mapped into a process segment or SEGMAP. Generally the difference between this and the resident figure is what is on the free list associated with this file.

- Shared – The amount of memory that this file has in memory that is shared with more than one process.
- Pageins – The amount of minor and major pageins for this file.
- Pageouts – The amount of pageouts for this file.
- Filename – The filename of the vnode or if not known, the device and inode number in the format 0x0000123:456.

The GUI will only display the largest 250 files. A status panel at the top of the display shows the total amount of files and the number that have been displayed.

The second mode of the MemTool GUI is the Process Memory display. Click on the Process Memory checkbox to select this mode.

PID	Virtual	Resident	Shared	Private	Process Name
10186	24920k	21072k	1776k	19296k	/usr/dist/share/framemaker,v5
9875	145288	146641	1328k	13336k	/usr/openwin/bin/Xsun :0 -no
11723	153601	127361	2568k	10168k	/opt/RMCmem/bin/5.7/sparcv
11768	7312k	6816k	1816k	5000k	xv –crop 18 62 740 530
10004	9152k	5560k	3424k	2136k	dtwm
11657	104321	5456k	3464k	1992k	/usr/dt/bin/dtmail
10800	309361	4192k	2216k	1976k	/usr/dist/share/netscape,v4.51
10010	7016k	4424k	3368k	1056k	/usr/dt/bin/dtterm –session d
10390	6992k	4392k	3376k	1016k	/usr/dt/bin/dtterm
10807	191361	3104k	2216k	888k	(dns helper)
158	3336k	2192k	1416k	776k	/usr/lib/autofs/automountd
10062	1544k	1152k	744k	408k	/bin/csh
258	2304k	1768k	1384k	384k	milibsa –r –p 32825

Figure 6-2 MemTool GUI – Process Memory

The Process Memory display (Figure 6-2), shows the process table with a memory summary for each process. Fields are described as follows:

- PID – Process ID of the process.
- Virtual – The virtual size of the process, including swapped-out and unallocated memory.
- Resident – The amount of physical memory that this process has, including shared binaries, libraries, and so forth.
- Shared – The amount of memory that this process is sharing with another process, such as shared libraries, shared memory, and so forth.
- Private – The amount of resident memory that this process has which is not shared with other processes. This figure is essentially Resident - Shared and does not include the application binaries.
- Process – The full process name and arguments.

The third mode shows the Process Matrix which is the relationship between processes and mapped files. Click on the Process Matrix checkbox to select this mode.

Display Type		Displaying		Show		Selection Totals			
		Processes	out of	All/Filt	Iter	Private	44872	Text/Library	13440
		Files	out of	Selected		Total	58312		
Private	21177933885832940	334467708	10186	2675	11723	14957	10004	40800	9097
	2119422658883424	334467708	9600k	10000k	9600k	1712k	2112k	1112k	1312k
	21191815068334564	334210028							
libXvbe.so.4	libXvbe.so.4	1450k			1472k	1472k			1472k
libc.so.1	libc.so.1	624k	622k	592k	622k	592k	622k	622k	622k
libX11.so.4	libX11.so.4	440k	440k	440k	440k	440k	440k	440k	440k
libstdc.so.1	libstdc.so.1	424k	422k	422k	424k	424k	424k	422k	422k
MM.so.2	MM.so.2	360k		360k	360k		360k	360k	360k
MM.so.4	MM.so.4	336k		336k	336k		336k	336k	336k
MMISvc.so.1	MMISvc.so.1	280k		288k	288k		288k	288k	288k
M.so.1	M.so.1	120k	128k	120k	120k	120k	120k	120k	120k
MDTWidget.so.2	MDTWidget.so.2	130k			128k	120k			120k
MDTServer.so.1	MDTServer.so.1	112k			112k	112k			
MDTTerms.so.2	MDTTerms.so.2	96k							96k

Figure 6-3 MemTool GUI – Process Matrix

Across the top of the Process Matrix display (Figure 6-3), is the list of processes that were viewed in the Process Memory display. Down the side is a list of the files which are mapped into these processes.

Each column of the matrix shows the amount of memory mapped into that process for each file, with an extra row for the private memory associated with that process.

The matrix can be used to show the total memory usage of a group of processes. By default, the summary box at the top right corner shows the memory used by all of the processes displayed.



Things You Can Tune

- Page daemon parameters
 - *maxpgio*
 - *lotsfree,minfree,desfree*
 - *fastscan,slowscan*
- Priority paging
- Amount of physical memory
- Swap space placement
- */tmp* usage
- Use of shared libraries

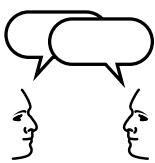
Things You Can Tune

The list in the overhead above contains the most commonly adjusted tuning parameters for the memory subsystem. In addition, the list provides some actions that you can take to improve the efficiency of your memory system.

With priority paging enabled, the file system scan rate will be higher. More pages need to be scanned to find file pages that it can steal (process private memory and executable pages are skipped). High scan rates are usually found on systems with heavy file system usage and should not be used as a factor for determining memory shortage.

If you have the Solaris 7 OS (or the appropriate patches for 2.6 and 2.5.1), the `memstat` command will reveal whether you are paging to the swap device and if you are, the system is short of memory.

If you have high file system activity, you will find that the scanner parameters are insufficient and will limit file system performance. To compensate for this, set some of the scanner parameters to allow the scanner to scan at a high enough rate to keep up with the file system.



By default, the scanner is limited by the *fastscan* parameter, which reflects the number of pages per second that the scanner can scan. It defaults to scan a quarter of memory every second and is limited to 64 Mbytes/sec. The scanner runs at half of *fastscan* when memory is at *lotsfree*, which limits it to 32 Mbytes/sec. If only one in three physical memory pages are file pages, the scanner will only be able to put 11 Mbytes ($32 \div 3 = 11$ Mbytes) of memory on the free list, thus limiting file system throughput.

You need to increase *fastscan* so that the page scanner works faster. A recommended setting is one quarter of memory with an upper limit of 1 Gbyte per second. This translates to an upper limit of 131072 for the *fastscan* parameter. The *handspreadpages* parameter should also be increased with *fastscan* to the same value.

Another limiting factor for the file system is *maxpgio*. The *maxpgio* parameter is the maximum amount of pages the page scanner can push. It can also limit the amount of file system pages that are pushed. This in turn limits the write performance of the file system. If your system has sufficient memory, set *maxpgio* to something large, such as 65536. On E10000 systems this is the default for *maxpgio*.

Remember, poorly performing memory subsystem can cause problems with CPU performance (from excessive scanning) and the I/O subsystem (unnecessary I/O operations).

If you have memory problems, work on them first. You may eliminate or reduce many other problems seen in the system that result from the memory subsystem.

Check Your Progress

Before continuing on to the next module, check that you are able to accomplish or answer the following:

- Describe the structure of virtual memory
- Describe how the contents of main memory are managed by the page daemon
- Understand why the swapper runs
- Interpret the measurement information from these components
- Understand the tuning mechanisms for these components

Think Beyond

Consider the following:

- Why is the use of main memory modeled so closely after a cache?
- What is the benefit of managing virtual memory in segments?
- If the memory subsystem is performing poorly, where else would you see performance problems? Why?
- Why are tuning parameters other than the scan rate generally not very useful?

Objectives

Upon completion of this lab, you should be able to:

- Use the `pmap` command to view the address space structure
- Identify the contents of a process address space
- Describe the effect of changing the `lotsfree`, `fastscan`, and `slowscan` parameters on a system
- Recognize the appearance of memory problems in memory tuning reports

For the Solaris 7 environment in 64-bit mode: All tuning parameters are now 64-bit fields. To change them with `adb`, use the following two steps:

1. For `adb`, substitute the following verbs

X (32 bit) to J (64 bit)

D (32 bit) to E (64 bit)

W (32 bit) to Z (64 bit)

2. Rename `paging_input_64` to `paging_input`.

Tasks

Process Memory

Complete the following steps:

1. Using the ps command or the /usr/proc/bin/ptree command, locate the process identifier (PID) of your current session's Xsun process.
-

2. Run:

```
# /usr/proc/bin/pmap -x Xsun-pid
```

How much real (resident) memory is the Xsun process using?
(Totals are in Kbytes.)

How much of this real memory is shared with other processes on the system?

How much private (non-shared) memory is the Xsun process using?

Page Daemon Parameters

Determining Initial Values

Use the `lab6` directory for these exercises.

1. Observe the effects of modifying various kernel paging parameters on a loaded system. The first step is to dump all of the current paging parameters.

Type:

`./paging.sh`

Record the output.

`lotsfree`_____ pages

`desfree`_____ pages

`minfree`_____ pages

`slowscan`_____ pages

`fastscan`_____ pages

`physmem`_____ pages

`pagesize`_____ bytes

2. Convert `lotsfree` to bytes from pages. You will use these values later in this lab.

Divide this number by 1024 to get `lotsfree` in Kbytes.

Changing Parameters

In this portion of the lab, you will put a load on the system by running a program called `mem_hog`, which locks down a portion of physical memory. Then you will run a shell script called `load3.sh`, which puts a load on the system's virtual memory subsystem. This script should force the system to page and, perhaps, swap. You will run `mem_hog` and `load3.sh` several times, each with a different set of paging parameters to observe their effect.

1. Observe the system under load with the default parameters.

```
vmstat -S 5
```

2. In another terminal window, type:

```
prtconf
```

to obtain the amount of physical memory on your machine.

3. Type:

```
./mem_hog physical_memory
```

where `physical memory` is the total number of Mbytes of physical memory, less four, on your machine; that is, `mem_hog 24`, if your machine has 28 Mbytes of physical memory.

Note – When `mem_hog` has locked down its memory, it prompts you to press Return to exit. This frees up any memory that has been locked, so *do not* press Return yet.

4. In a third window, type:

```
./load3.sh
```

This puts a load on the system. It takes about 3 to 5 minutes to run.

Note – `load3.sh` produces an “invalid file type” error as well as other job completion messages; this is normal. Ignore the error.

Some things to observe in `vmstat` while the script is running are:

- ▼ `freemem` should hover around the value of `lotsfree`. (Remember, `freemem` is in Kbytes.)
- ▼ Scan rate and number of Kbytes being freed per second. Most of this activity is due to the page daemon.
- ▼ Disk activity.
- ▼ Time spent in user and system mode.
- ▼ Data that is being paged in and paged out.

When `load3.sh` is finished, press Control-c to exit `vmstat`.

5. Record the same data in a `sar` file. Type:

```
sar -A -o sar.norm 5 60
```

6. In another window, type:

```
./load3.sh
```

Wait for `load3.sh` to complete. If `sar` is still running, use Control-c to stop the output of `sar`.

7. In the `mem_hog` window, press Return to exit `mem_hog`. You can leave `mem_hog` running if it does not impact your system performance too much.
8. In a new window, type:

```
adb -kw
```

9. Try increasing `lotsfree` to one-tenth of main memory. In `adb`, type:

```
lotsfree/W 0tnew_lotsfree
```

(Remember, the value you use for `new_lotsfree` should be in pages.)

10. To verify, in `adb`, type:

```
lotsfree/D
```

You should see the value you just set.

11. Increase slowscan to half of fastscan. In adb, type:

```
slowscan/W 0tnew slowscan
```

12. To verify the new value for slowscan, type:

```
slowscan/D
```

Note – Type \$q to exit adb. For this lab, you may want to keep this window open.

Given these alterations, what sort of behavior do you expect to see on the system with respect to the following:

▼ Amount of time the CPU spends in user and system mode

▼ Average amount of free memory on the system.

▼ Number of pages being scanned and put on the freelist.

13. If mem_hog is not still running, invoke mem_hog by typing:

```
./mem_hog physical_mem
```

14. In another window, type:

```
sar -A -o sar.1 5 60
```

15. In another window, type:

```
./load3.sh
```

Wait for load3.sh to complete.

16. Exit mem_hog.

17. In one window, type:

```
sar -f sar.norm -grqu
```

18. In another window, type:

```
sar -f sar.1 -grqu
```

Did you see the changes you expected? Explain why or why not.

19. Restore lotsfree to its original value. In the adb window, type:

```
lotsfree/W 0toriginal_value_of_lotsfree
```

Scan Rates

To examine the scan rates:

1. Lower slowscan. In the adb window, type:

```
slowscan/W 0t10
```

2. Lower fastscan. In the adb window, type:

```
fastscan/W 0t100
```

What effect do you think this will have on the system's behavior with respect to:

▼ Number of pages scanned per second

▼ Number of pages free per second

▼ Average amount of free memory

▼ Size of the swap queue

3. If necessary, generate paging activity by running `mem_hog`. Type:

mem_hog *physical_mem*

4. In another window, type:

```
sar -A -o sar.2 5 60
```

5. In another window, type:

load3.sh

Wait for load3.sh to complete.

- ## 6. Exit mem_hog.

7. In one window, type:

```
sar -f sar.norm -gru
```

8. In another window, type:

sar -f sar.2 -grqu

Did you see the changes you expected? Explain why or why not.

9. Restore slowscan to its original value. In the adb window, type:

slowscan/W 0 to *original_value*

-
10. Restore fastscan to its original value. In the adb window, type:

fastscan/w 0t*original_value*

11. Exit adb using the **\$q** command

Objectives

Upon completion of this module, you should be able to:

- Define a bus
- Explain general bus operation
- Describe the processor buses: UltraSPARC™ Port Architecture (UPA), Gigaplane, and Gigaplane XB
- Describe the data buses: SBus and Peripheral Component Interconnect (PCI)
- Explain how to calculate bus loads
- Describe how to recognize problems and configure the system to avoid bus problems
- Determine dynamic reconfiguration considerations

Relevance

Discussion – The following questions are relevant to understanding the content of this module:



- What is the purpose of a bus?
- Where in the system are the buses?
- What could happen if a bus does not operate correctly?
- What symptoms could you see if a bus was overloaded?

Additional Resources



Additional resources – The following references can provide additional details on the topics discussed in this module:

- Cockcroft, Adrian, and Richard Pettit. 1998. *Sun Performance and Tuning*. Palo Alto: Sun Microsystems Press/Prentice Hall.
- Wong, Brian L. 1997. *Configuration and Capacity Planning for Solaris Servers*. Palo Alto: Sun Microsystems Press/Prentice Hall.
- Catanzaro, Ben. 1994. *Multiprocessor System Architectures*. SunSoft Press/Prentice Hall.
- *SPARC Architecture Reference Manual*.
- *Solaris Architecture and Internals*, Mauro/McDougall, Prentice Hall/SMI Press, ISBN 0-13-022496-0.
- Man pages for the `prtdiag` command.



What Is a Bus?

- It is a mechanism to transfer data, just like a network.
- It usually connects more than two components.
- It runs at a fixed speed.
- It is usually bidirectional and broadcasts messages.
- It is like a highway:
 - Speed limit is a maximum, not the required speed.
 - It has lower speed for "on and off ramps."
- Higher speeds mean more traffic capacity.

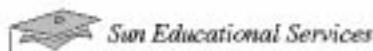
What Is a Bus?

A bus is conceptually a cable, connecting two or more components, like a network. The bus may actually be implemented as a cable, as traces on a circuit board, as a wireless link, or some combination. The common thing is that all transfer data and control signals from one location to another.

Almost every data transfer mechanism in the system has the characteristics of a bus: a fixed or maximum bandwidth, one active transmitter at a time, a contention resolution scheme, and bi-directional transfer ability.

Very high-speed buses may be serviced and supported by slower buses, much as an asynchronous transfer mode (ATM) network may be de-multiplexed into multiple 10baseT or 100baseT links.

With so many buses, which have so many common characteristics, if the basic principles of a bus are understood they can be applied to many different situations.



General Bus Characteristics

- Circuit switched (SBus)
 - Operates like a telephone
 - Keeps the bus busy for the whole operation
- Packet switched (Gigaplane and PCI buses)
 - Operates like a network
 - Sends a request and then frees the bus for the next request

Most buses are packet switched.

General Bus Characteristics

There are two major categories of buses: circuit switched and packet switched.

A circuit-switched bus operates like a telephone, where a connection must be made at each end, and the bus remains busy for the entire operation. Once a transaction is initiated, it must be rejected or completed before any other transaction may be started. Circuit-switched buses are generally easier to implement, but have significantly slower throughput in most applications.

Packet-switched buses operate like a network, where a message is sent to a specific destination requesting an operation, and then the bus is released to other traffic. When the requested operation is complete, the remote destination sends a message (or messages) back with the results of the operation.

Most buses today are packet switched for performance reasons even though they are more complicated to implement.



System Buses

- Are used to connect sections of the system to each other
 - They include CPUs, peripheral controllers, and memory.
 - Provide many different kinds of system buses, depending on cost, technology, and needs
 - Are simpler on smaller systems
 - Cost issues are important.
 - For large systems throughput outweighs the cost

System Buses

The system bus is used to connect various sections of the hardware together, such as system boards to other system boards, or CPUs to memory banks.

There are many different kinds of system bus, depending on cost and performance requirements. Some systems have multiple layers of buses, with slower buses connecting to faster buses, to meet performance, flexibility, design, and cost requirements.

System buses change and evolve over the years; Sun has used many different buses in its SPARC systems. Each new processor family generally uses a new system bus.



System Buses: MBus and XDbus

- MBus – For small desktop multiprocessors (sun4m)
 - Is circuit switched
 - Is a separate I/O bus from CPU and memory
 - Looks like a current Pentium system design
- XDBus – SS1000, SC2000, and CS6400 (sun4d)
 - Is packet switched
 - Has 1 on 1000, 2 on 2000, and 4 on 6400
 - Is easy to overload with a full configuration

MBus and XDbus

The MBus (module bus) was developed for the first Sun SPARC multiprocessor systems, the Sun4/600 series, and was also used in desktop uniprocessor and multiprocessor systems. These systems made up the sun4m architecture family.

The XDBus (Xerox databus) was used in Sun's first generation, large, datacenter systems, the sun4d architecture family. Unlike the general purpose systems, the family members had different numbers of parallel buses to provide additional bandwidth without the need for faster or wider buses.



System Buses: UPA and Gigaplane Bus

- UPA – Used by the Ultra™ systems
 - Is a crossbar which connects CPUs, memory, and I/O
 - Is very efficient and very fast
- Gigaplane – Used in the Ultra servers
 - Connects multiple UPA buses (on system boards)
 - Is configurable while you are running the system
 - Is what dynamic reconfiguration (DR) does

UPA and Gigaplane Bus

In the Sun UltraSPARC family, two new system buses were provided.

The UPA bus is a crossbar (all components directly connected) bus designed to connect a few elements together. It is used as the main system bus in the UltraSPARC desktop systems. It is very fast and very efficient.

The Gigaplane bus was introduced with the Enterprise X000 family of systems. In these servers, each system board has a UPA bus which connects the board's components to a Gigaplane backbone bus. The Gigaplane bus has the ability to electrically and logically connect and disconnect from the UPA buses while it is running, providing the ability for the X000 and X500 servers to do dynamic reconfiguration (the ability to add and remove system boards from a running system).



Gigaplane XB Bus

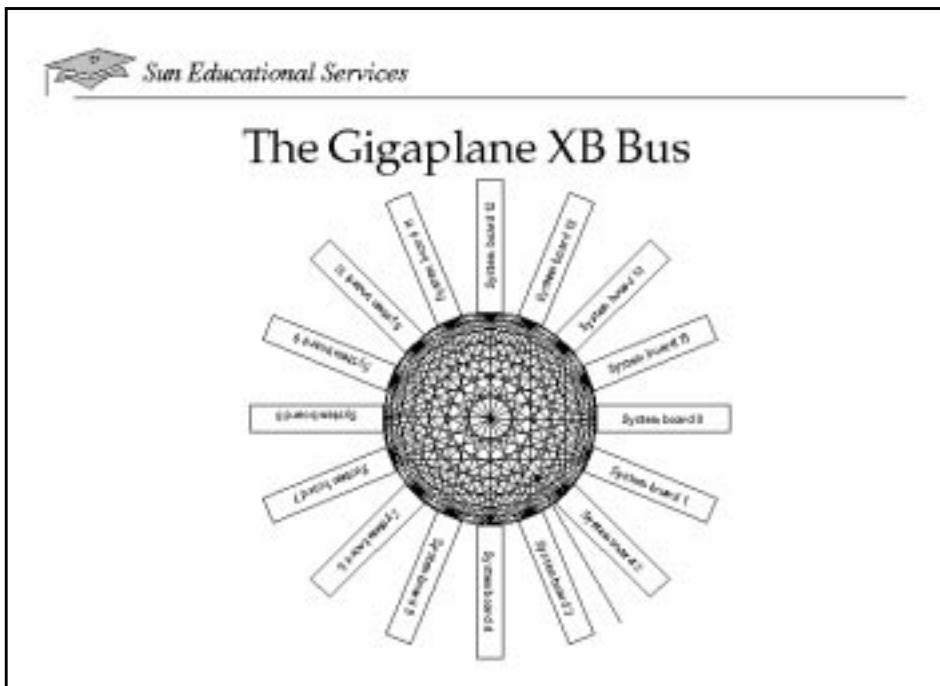
- Is only on the Sun Enterprise 10000
- Is based on the Gigaplane bus
- Has point-to-point connections to all 16 boards
- Allows addition and removal of whole boards from domains
- Contains multiple parallel sub-buses
 - Can run without most of them
- Runs at 100 MHz
 - Can do two operations per cycle

Gigaplane XB Bus

The Gigaplane XB (crossbar) bus is an extension of the Gigaplane bus, which is currently used only in Enterprise 10000 systems. The Gigaplane XB bus consists logically of many individual Gigaplane buses, connected point-to-point between the 16 system boards in the E10000. This provides the appearance of a Gigaplane bus dedicated to connecting each pair of system boards.

Just as with the Gigaplane bus, the Gigaplane XB bus provides the ability to perform dynamic reconfiguration.

In addition, the Gigaplane XB is constructed in a modular fashion that allows the system to continue to operate if sections of it have failed.



The Gigaplane XB Bus

This diagram shows the crossbar connections between each system board in the Enterprise 10000's Gigaplane XB bus. Each line represents a logically separate Gigaplane bus.

Sun System Backplanes Summary

Characteristics of recent Sun system backplanes are summarized in Table 7-1.

Table 7-1 System Backplane Characteristics.

System	Bus	Speed, Width	Bandwidth Burst (Sustained)
SPARCstation 10 Model 20	MBus	33 MHz, 64 bits	264 MB/sec (86 MB/sec)
SPARCstation 10 Model 30	MBus	36 MHz, 64 bits	288 MB/sec (94 MB/sec)
SPARCstation 10, SPARCsystem 600MP	MBus	40 MHz, 64 bits	320 MB/sec (105 MB/sec)
SPARCstation 20	MBus	50 MHz, 64 bits	400 MB/sec (130 MB/sec)
SPARCstation 20 Model 514	MBus	40 MHz, 64 bits	320 MB/sec (105 MB/sec)
SPARCserver 1000	XDBus	40 MHz, 64 bits	320 MB/sec (250 MB/sec)
SPARCcenter 2000	XDBus	40 MHz, 64 bits (* 2)	640 MB/sec (500 MB/sec)
SPARCserver 1000E	XDBus	50 MHz, 64 bits	400 MB/sec (312 MB/sec)
SPARCcenter 2000E	XDBus	50 MHz, 64 bits (* 2)	800 MB/sec (625 MB/sec)
Cray SuperServer 6400	XDBus	55 MHz, 64 bits (* 4)	1.8 GB/sec (1.4 GB/sec)
Ultra-1/140	UPA	72 MHz, 128 bits	1.15 GB/sec (1.0 GB/sec)
Ultra-1/170	UPA	83.5 MHz, 128 bits	1.3 GB/sec (1.2 GB/sec)
Ultra-2/2200	UPA	100 MHz, 128 bits	1.5 GB/sec (1.44 GB/sec)
Ultra Enterprise x000	Gigaplane/ UPA	83.5 MHz, 256 bits	2.6 GB/sec (2.5 GB/sec)
Ultra Enterprise 10000	GigaplaneX B/UPA	100 MHz	12.8 GB/sec



Peripheral Buses

- Connect peripherals to the system buses
- Are slower than the system buses, usually by a lot
- Are circuit or packet switched
- Can easily be the (hidden) limiting factor in I/O performance
- Are provided as PCI or SBus by Sun
 - You can mix PCI and SBus in Enterprise servers.
 - All new systems are PCI.

Peripheral Buses

I/O devices generally run at a much lower bandwidth than the system bus. Trying to connect them to the system bus would be a very expensive process. In support of this, many different peripheral buses have evolved to connect I/O devices, usually through interface cards, to the system bus.

These buses can be either circuit or packet switched, depending on cost and performance considerations. These buses, not usually even seen, can be a hidden bandwidth limiter on the system, and can indeed cause not only performance problems but the appearance of hardware problems as well.

Sun now provides two different peripheral interconnect buses: the SBus and the PCI bus. Server systems may contain a mixture of both. All new systems from Sun, both desktop and server, feature PCI as the only or the preferred peripheral interconnect bus.



SBus Versus PCI Bus

- SBus runs at 20–25MHz, 32 and 64 bits wide.
 - Its peak bandwidth is 200 Mbytes per second.
- PCI bus runs at 33 or 66 MHz and 32 or 64 bits wide.
 - Peak bandwidth is 528 Mbytes per second.
 - It is provided on all new Sun hardware.
- There are many 100-Mbyte interface cards today.
 - SOC+, SCI, ATM OC-12, gigabit Ethernet, and so on are available.

SBus Versus PCI Bus

The SBus is a 25-MHz circuit switched bus with a peak theoretical total bandwidth of around 200 Mbytes/sec in its fastest implementations. There are many SBus cards available today to perform most functions required by a Sun system.

The PCI bus was originally seen in personal computer (PC) systems. The PCI bus can run at 32- and 64-bit data widths, and at 33-, 66-, and 100-MHz clock rates. This provides a maximum theoretical total bandwidth for a 64-bit, 66-MHz card of over 500 Mbytes/sec. With hundreds of existing PCI cards, support in Sun PCI system requires only a device driver.

There are several 100-Mbytes/sec I/O interface cards available today, the appeal of the PCI bus is obvious.

Enterprise Server I/O Boards

There are now five types of Sun Exx00 server I/O board, each with different configuration characteristics.

- Type 1 – Three SBus slots and two onboard SOC (serial optical channel) FC/OMs (Fibre Channel optical modules)
- Type 2 – Two SBuses, one UPA, and two onboard SOC FC/OMs
- Type 3 – Four PCI channels and two (short) PCI slots
 - ▼ No SCSI interface
- Type 4 – Three SBuses and two onboard SOC+ GBICs (gigabit interface connectors)
- Type 5 – Two SBuses, one UPA, and two onboard SOC+ GBICs



PIO and DVMA

- PIO cards require the CPU to perform all data transfer.
 - The CPU can only move 8 or 64 bytes at a time.
 - They use lots of CPU resource and overhead.
 - They are used for lower performance or slow cards.
- DVMA bypasses the CPU and goes directly to memory.
 - It is a much more complicated setup.
 - The breakeven point with PIO is at about 2048 bytes.
 - Many cards can do both.
 - The driver chooses which is used (usually silently).

PIO and DVMA

Once the peripheral devices have been attached to the system, the system must be able to transfer data to and from them. Mediated by the device driver, this is done over the system bus which is connected to the peripheral bus.

The exact mechanisms of these transfers are beyond the scope of this course, but the essential feature of the transfers is that the peripheral card appears to be resident in physical memory; that is, writes to a section of what looks like main memory are actually writes to the interface card.

This connection allows two types of data transfer to be performed between the peripheral card, representing the device, and the system. These are programmed I/O (PIO) and direct virtual memory access (DVMA).

PIO

Programmed I/O forces the CPU to do actual copies between the device memory and main memory. Moving 8 bytes (64 bytes on UltraSPARC systems) at a time, the CPU copies the data from one location to the other. This can be very CPU intensive for a high-speed device, and limits the bandwidth at which the device can operate to the ability of the CPU to move the data.

PIO is quite easy to use in a device driver, and is normally used for slow devices and small transfers.

DVMA

DVMA allows the interface card, on behalf of the device, to transfer data directly to memory, bypassing the CPU. DVMA is capable of using the full device bandwidth to provide much faster transfers. DVMA operations, however, require a much more complicated setup than PIO. The break-even point is at about 2000 bytes in a transfer.

Not all I/O cards support DVMA, and if they do, the driver may not have implemented support for it.

Which Is Being Used?

In most cases, it is not possible to tell. The driver implements either mechanism it chooses. In drivers that support both, it may use PIO in older, slower machines and DVMA in newer, faster machines. Usually no message is issued.

This ability to switch explains how a 100-Mbyte interface card, such as a scalable coherent interconnect (SCI) card (used with clustering) can operate on a 40-Mbyte/sec SBus. PIO is used to lower the transfer rate to what the system can tolerate (by definition). Throughput calculations made assuming that the full bandwidth is available will be overestimated in these cases, usually by a factor of 2 or 3.



prtdiag

- Is provided for Sun's sun4d and sun4u systems
 - SS1000 and SC2000 are sun4d systems.
 - UltraSPARC™ systems are sun4u systems.
- Shows the hardware configuration of the system
 - It is used to locate configuration problems.
- Requires knowledge of system part identification
 - The *Field Engineer's Handbook* should be used.
- Is similar to what prtconf does, but is easier to read

prtdiag

The prtdiag utility, first provided with the SunOS 5.5.1 operating system, provides a complete listing of a system's hardware configuration, up to the I/O interface cards.

It is supported for the sun4d and sun4u architecture systems, and is key to locating hardware configuration problems. Intended for use by people familiar with Sun's hardware nomenclature, you may need a reference such as the *Field Engineer's Handbook* to identify some of the interface cards completely.

You can get similar information from the prtconf command, but it is much harder to read and interpret.

**The prtdiag command is found in the
`/usr/platform/sun4[du]/sbin` directory or the
`/usr/platform/sun4u1/sbin` directory on the Sun Enterprise 10000.**



prtdiag CPU Section

```
# /usr/platform/sun4u1/sbin/prtdiag
System Configuration: Sun Microsystems sundu SUNW,Ultra-Enterprise-10000
System clock frequency: 83 MHz
Memory size: 1024 Megabytes
```

```
----- CPUs -----
Brd CPU Module Run Cache CPU CPU
    MHz MB Impl. Mask
-----
0   0   0     250  1.0 US-II 1.1
0   1   1     250  1.0 US-II 1.1
0   2   2     250  1.0 US-II 1.1
0   3   3     250  1.0 US-II 1.1
1   4   0     250  1.0 US-II 1.1
1   5   1     250  1.0 US-II 1.1
1   6   2     250  1.0 US-II 1.1
1   7   3     250  1.0 US-II 1.1
```

prtdiag CPU Section

The first section of the prtdiag report describes the CPUs present in the system.

This report, taken from an Enterprise 10000 system, shows eight CPUs on two system boards. The CPUs are 250-MHz UltraSPARC-II systems with 1 Mbyte of external cache. (The mask version is the revision level of the processor chip.)

 Sun Educational Services

prtdiag Memory Section

```
----- Memory -----
Memory Units: Size
 0: MB  1: MB  2: MB  3: MB
-----
Board 0      256      256      0      0
Board 1      256      256      0      0
```

- You can determine memory interleaving from this.
 - Proper interleaving can provide a 10 percent or more performance boost.
 - Systems configure this automatically and silently.
 - Some systems can do up to 16-way interleaving.

prtdiag Memory Section

The next section of the `prtdiag` report shows the physical memory configuration. This report shows two system boards, each with two banks of 256 Mbytes each, for a total system memory size of 1 Gbyte.

There are four memory banks possible on the system board, according to the report, but only two are filled. This usually allows what is called two-way memory *interleaving*.

Memory Interleaving

When writing, say, 64 bytes to memory (the normal transfer or line size), the system is not able to write it all at once. For example, a memory bank may have the ability to accept only 8 bytes at a time. This means that eight memory cycles or transfers will be required to write the 64 bytes into the bank.

If you could spread the data over two memory banks, you could transfer all of the data in four memory cycles, taking half the time required by one bank. If you could use four, it would take only two transfers. If there was a way to use eight banks, it would take only one cycle to move the 64 bytes to or from memory.

While four 64-byte requests would take the same amount of transfer time, interleaved or not, interleaving does provide significant performance benefits.

Aside from the shorter elapsed time, requests are always spread across multiple banks, providing a sort of memory RAID (redundant array of independent disks) capability. This helps avoid hot spots (over utilization) of a particular bank if a program is using data in the same area consistently. For some types of memory-intensive applications, increased speeds of 10 percent or more have been seen, just by using interleaving.

Systems configure interleave silently. Without knowing the characteristics of a system, you cannot always be sure that it is interleaving properly. For example, the desktop maintenance manuals specify the sequence that memory SIMMs (single in-line memory modules) should be installed in the system. The installation order is not always in physical order on the board. While the memory will still be used (in most cases), there will be no interleave. It is not always obvious what the proper installation order should be.

Some systems, when properly configured, are capable of 16-way (or more) interleave.

Check your systems to ensure that the memory has been properly installed for maximum interleave, and remember to take into account the possibility of interleave when configuring a system.

 Sun Educational Services

prtdiag I/O Section

I/O Cards					
Bus	Type	Freq	Slot	Name	Model
Bus	Type	Freq	Slot	Name	Model
0	SBus	25	0	qec/qe (network)	S2NN, S2S-3198
0	SBus	25	0	SUNW, soc/SUNW, plm	S01-2069
0	SBus	25	1	QLOG, lsg/sd (block)	QLOG, ISP1000
1	SBus	25	0	qec/qe (network)	S2NN, S2S-3198
1	SBus	25	0	SUNW, soc	S01-2069
1	SBus	25	1	QLOG, lsg/sd (block)	QLOG, ISP1000

- You can calculate bus loading based upon this output.
- You can move things if the bus is overloaded.

prtdiag I/O Section

The prtdiag I/O section shows, by name and model number, the interface cards installed in each peripheral I/O slot.

By using the information in this section, you can calculate the requested bandwidth for the bus to determine if you are causing I/O bottlenecks by installing interface cards with more bandwidth capability than the peripheral bus can handle.



Bus Limits

- A bus is like a freeway:
 - More traffic = more congestion
 - Too much traffic, things back up
 - Things get delayed, you get problems
 - Going faster takes more skill and money
- Bus saturation causes problems.
- You do not get any specific messages.

Bus Limits

A bus is like a freeway: it has a speed limit. Unlike a freeway, however, the bus speed cannot be exceeded.

The problem is that as you add more traffic, the bus tends to get congested and not operate as well. When the bus reaches capacity, no more traffic can flow over it. Any requests to use the bus must wait until there is room, delaying the requesting operation.

When this occurs, you do not get any specific messages; the system does not measure or report on this condition.



Bus Bandwidth

- If I have checks, I must have money.
- If I have slots, I must have bandwidth.
- Both might be wrong.
- Symptoms of this are:
 - Bad service time
 - Dropped network packets
 - Intermittent hardware problems
 - Irregular but constant problems

Bus Bandwidth

There is an old joke that goes, “If I have checks, I must have money.” The connection between a check and real money is lost. The same situation can occur with bus bandwidth: “If I have slots, I must have bandwidth.” Like checks and money, these two are not necessarily related.

If you have installed more cards into a peripheral bus than you have bandwidth for, some of the I/O requests will be delayed. The greater the delay, the more problems will become evident.

Problems from this situation, such as bad I/O response times, dropped network packets, and so on, occur intermittently but consistently, almost invariably during the busiest part of the day.



Diagnosing Bus Problems

- If after every board has been replaced, the problem is still there:
 - Check your configuration using prtdiag.
 - Correlate failures with system usage.
 - Trust the specifications, not the sales person.

Diagnosing Bus Problems

With these sorts of problems, it is not uncommon to have the service people replace every I/O board, system board, CPU, memory bank, and power supply, sometimes multiple times, and still have the problem persist. This might be a hint that this is not a hardware problem.

Check your configuration with prtdiag. Given the maximum bus bandwidths (see Appendix A) for your system, and the bandwidths necessary for the interface cards (also in Appendix A), it should be easy to tell if you have exceeded the available bandwidth for the peripheral bus.

Just because the slot is available does not mean that it can be used. If it seems unreasonable to support certain interface cards in the same bus, check the configuration. These problems are not widely known, and it is possible for both the sales and support staff to overlook a problem.



Avoiding the Problem

- Know how your system works
 - There are plenty of good white papers.
 - Be prepared to leave empty slots
 - Watch for unexpected problems
 - Understand the basics
 - Everything is either a bus or a cache.
 - Be sure to tune and monitor it accordingly.
 - Smarter is better than bigger.

Avoiding the Problem

When calculating throughput, realize that you will usually at best get 60 percent of the rated bandwidth of a peripheral device. It is not usually necessary to use actual transfer rates when checking a configuration; you will almost invariably get the same percentage of capacity either way.

To avoid the problem, follow the suggestions in the overhead image above. There are many ways to create the problem, but only a few to solve it.



Avoiding the Problem

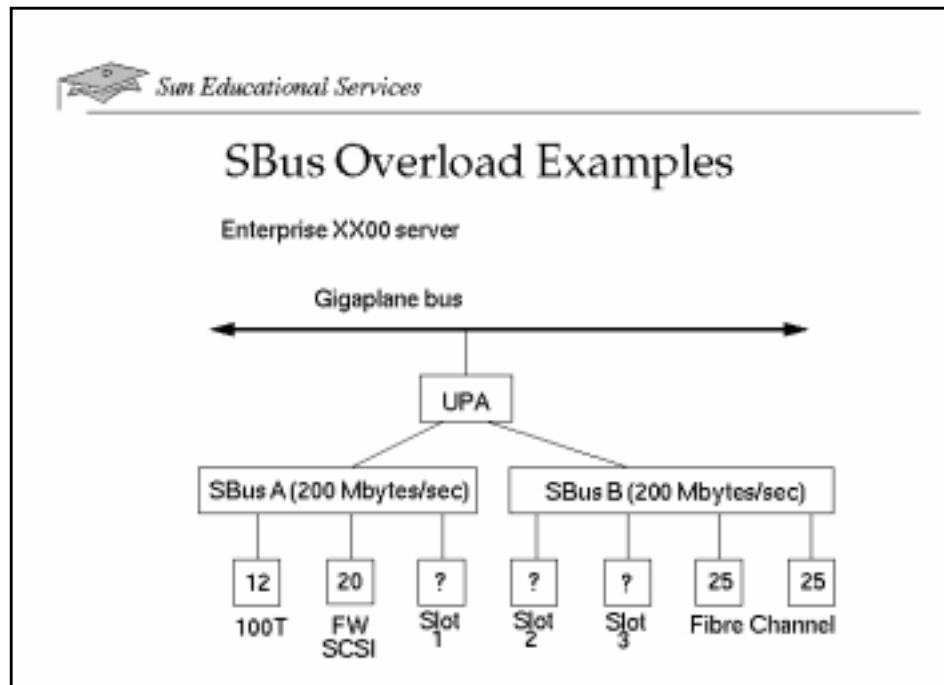
- Know how your system works
 - There are plenty of good white papers.
- Be prepared to leave empty slots
- Watch for unexpected problems
- Understand the basics
 - Everything is either a bus or a cache.
 - Be sure to tune and monitor it accordingly.
 - Smarter is better than bigger.

There is plenty of detailed information available about the capabilities and requirements of I/O interface cards and system capabilities. There are white papers on the specific systems and devices, and some excellent books.

Unexpected and unexplained problems (all the patches are on, all the hardware has been replaced) may be strong indications of configuration problems.

Remember the basics when tuning and configuring a system: almost everything is a bus or a cache and should be tuned accordingly. Monitor the system carefully, and try to understand what it is doing.

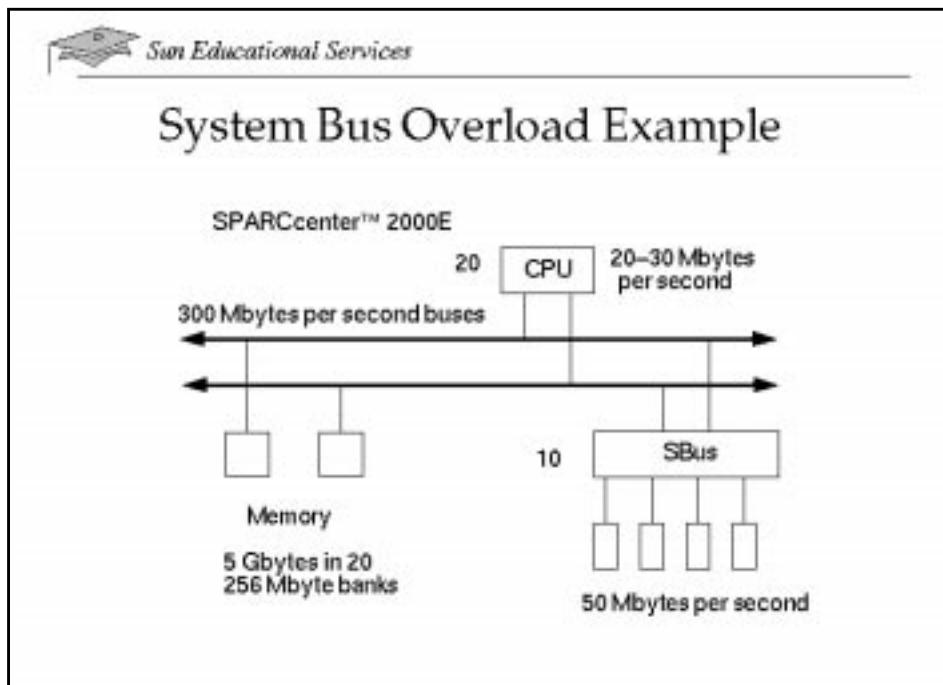
Be prepared to leave empty slots in the system. You may not have bandwidth for some of those slots.



SBus Overload Example

The overhead image above shows an example of a potentially overloaded SBus. Without knowledge of the SBus management of the server slots, it would be possible to put SOC+ (Serial Optical Channel+) cards (100 Mbytes/sec FC-AL [Fibre Channel-Arbitrated Loop] interfaces) into slots 2 and 3 and overload SBus B.

In this case, assuming that all onboard devices (those shown) are in use, the best choice would be to put the SOC+ cards in slots 1 and 2, and leave slot 3 empty.



System Bus Overload Example

This example shows the possible overloading of the *system bus*. Remember that the system bus has a fixed bandwidth, and it is possible to overconfigure a system such that the system bus itself becomes overloaded. (Table 7-2) shows how this is done.

Table 7-2 SC2000E System Bus Loads

Component	Quantity	Bandwidth	Total
CPU	20	25 Mbytes	500 Mbytes
SBus	10	50 Mbytes	500 Mbytes
Total		1000 Mbytes	

With a peak request for 1 Gbyte/sec bandwidth, and at most 600 Mbytes available, delays are inevitable for the system bus traffic as utilization grows. Unfortunately, this kind of overload often appears as watchdog resets. Hardware is often replaced when the configuration is at fault.



Dynamic Reconfiguration Considerations

- Watch for problems which occur when you add boards, not when you remove them
- Make sure you have bandwidth when you add components
- Be careful replacing slower cards or components
- Consider reorganizing to spread the load
 - Be careful of controller and component renumbering
- Plan ahead

Dynamic Reconfiguration Considerations

Dynamic reconfiguration (DR) enables you to add and remove I/O boards (Enterprise xx00 servers) and system boards (Enterprise 10000) in a running system.

New configuration performance problems will not occur when you remove boards; you are lessening the demand for system resource. Problems can occur when you add boards.

Follow the guidelines given in the overhead image above when you add boards or use DR to add new components to existing boards. Make sure that you have the bandwidth, in the peripheral and system buses, to support the components you are adding.

Consider reorganizing the existing components and boards to provide better performance for the new components. If you know that you will be adding components at a later time, plan for it now to avoid problems then.



Tuning Reports

- Not much is available in this area.
- Vigilance is the best prevention.
- When you add new cards, be sure to check prtdiag.
- Remember that empty slots do not mean available bandwidth.
- Strange behavior in tuning reports might be related to bus problems.
 - It usually occurs when the system is most heavily loaded.

Tuning Reports

There is very little information available in this area from tuning reports, largely because system configuration is almost always static, which is not interesting to a monitor. Monitors do not even report on the current configuration when they start.

The best protection here is using prtdiag on current systems, and doing a bandwidth calculation before adding boards to an existing system. Keep in mind:

- An empty slot does not imply a usable slot.
- Strange behavior, especially behavior that continues after hardware has been replaced, may be due to configuration problems.
- Failures due to configuration problems almost invariably appear when the system is the most heavily loaded.

Check Your Progress

Before continuing on to the next module, check that you are able to accomplish or answer the following:

- Define a bus
- Explain general bus operation
- Describe the processor buses: UPA, Gigaplane, and Gigaplane XB
- Describe the data buses: SBus and PCI
- Explain how to calculate bus loads
- Describe how to recognize problems and configure the system to avoid bus problems
- Determine dynamic reconfiguration considerations

Think Beyond

Consider these questions:

- Why are there so many different kinds of buses?
- Why does not every new system from Sun use the Gigaplane XB bus?
- Why do some systems use several different buses to connect their components?
- How could you tell if a device was using PIO or DVMA?
- How could you confirm whether you had a hardware or a configuration problem?

Objectives

Upon completion of this lab, you should be able to:

- Identify interface board characteristics
- Determine optimum system configuration
- Assess alternate server configurations

Tasks

System Configuration

Complete this step:

1. Using the following `prtfdiag` output and the tables in Appendix A, determine the best configuration for the following system:

```
# /usr/platform/'arch -k'/sbin/prtdiag
System Configuration: Sun Microsystems sun4d SPARCserver 1000E
System clock frequency: 50 MHz
Memory size: 128Mb
Number of XDBuses: 1
      CPU Units: Frequency Cache-Size          Memory Units: Group Size
                  A: MHz MB    B: MHz MB          0: MB     1: MB     2: MB     3: MB
      -----
Board0:        60 1.0      60 1.0          128       0       0       0
Board1:        60 1.0      60 1.0          0         0       0       0
=====SBus Cards=====
Board0:        SBus clock frequency: 25 MHz
            0: dma/esp(scsi)      'SUNW,500-2015'
            lebuffer/le(network) 'SUNW,500-2015'
            1: DOLPHIN,sci
            2: qec/be(network)    'SUNW,270-2450'
            3: SUNW,socal/sf(scsi-3)'501-2069'

Board1:        SBus clock frequency: 25 MHz
            0: dma/esp(scsi)      'SUNW,500-2015'
            lebuffer/le(network) 'SUNW,500-2015'
            1: DOLPHIN,sci
            2: qec/be(network)    'SUNW,270-2450'
            3: SUNW,socal/sf(scsi-3)'501-2069'

No failures found in System
=====
```

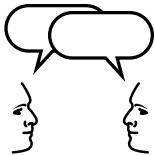
```
#
```

Objectives

Upon completion of this module, you should be able to:

- Demonstrate how I/O buses operate
- Describe the configuration and operation of a SCSI bus
- Explain the configuration and operation of a Fibre Channel bus
- Demonstrate how disk and tape operations proceed
- Explain how a storage array works
- List the RAID levels and their uses
- Describe the available I/O tuning data

Relevance



Discussion – The following questions are relevant to understanding the content of this module:

- What parts of I/O tuning are related to concepts you have already seen?
- What makes I/O tuning so important?
- Why must you understand your peripheral devices to properly tune them?

Additional Resources



Additional resources – The following references can provide additional details on the topics discussed in this module:

- Cockcroft, Adrian, and Richard Pettit. 1998. *Sun Performance and Tuning*. Palo Alto: Sun Microsystems Press/Prentice Hall.
- Wong, Brian L. 1997. *Configuration and Capacity Planning for Solaris Servers*. Palo Alto: Sun Microsystems Press/Prentice Hall.
- Ridge, Peter M., and David Deming. 1995. *The Book of SCSI*. No Starch Press.
- Benner, Alan F. 1996. *Fibre Channel*. McGraw-Hill.
- *Solaris Architecture and Internals*, Mauro/McDougall, Prentice Hall/SMI Press, ISBN 0-13-022496-0.
- Chuck Musciano, 0, 1, 0+1... *RAID Basics*, SunWorld, June 1999.
- Man pages for the `prtconf` command and performance tools (`sar`, `iostat`).



SCSI Bus Overview

- Was originally shipped in 1982; SCSI-1 in 1986
- Is standardized now at SCSI-3
 - Upward compatible evolution
- Supports disk, tape, network devices, media changers, CD-ROMs, and so on
- Is the most common computer peripheral connection
- Defines message traffic and bus operation
- Uses principles which apply to most I/O buses

SCSI Bus Overview

The SCSI bus was designed in the early 1980s to provide a common I/O interface for the many small computer systems then available. Each required its own proprietary interface, which made development of standard peripherals impossible. The SCSI bus helped solve this problem by providing a standard I/O interface.

The SCSI bus was originally shipped in 1982, and its initial standard (SCSI-1) was approved in 1986. The current level of standardization is SCSI-3, with further work continuing.

The SCSI standard specifies support for many different devices, not just disk and tape, but diskette drives, network devices, scanners, media changers, and others.

The SCSI bus, probably the most common peripheral interconnect today, is fully specified by the standards in terms of cabling, connectors, commands, and electrical signalling.

Conceptually, it operates like Ethernet, acting as a network between devices, although the implementation details are quite different. It is very similar to the IBM mainframe parallel channels in operation. Indeed, most I/O buses follow similar principles.



SCSI Bus Characteristics

- There are several types of SCSI buses.
- They are distinguished by their mix of characteristics.
 - Speed – How fast the signals move
 - Width – How much data moves at a time
 - Length – How long the bus can be
- Buses can support several different sets of characteristics concurrently.
- Device drivers negotiate a device's behavior on the bus.
- It behaves like a packet-switched bus.

SCSI Bus Characteristics

There are several different types of SCSI bus, distinguished by speed, length (distance it can support), and width (number of bytes transported at once).

A SCSI bus may be able to support many different sets of characteristics simultaneously, depending on the attached devices. The specific characteristics supported by a given device are negotiated with the attaching system's device drivers.

The SCSI bus operates like a packet-switched bus: it connects to the device and sends a message, then disconnects and waits for a response. The SCSI bus usually does not have to remain connected during an entire I/O operation, such as a read (although it can). This behavior is determined by the device.



SCSI Speeds

Name	Maximum Speed
Asynchronous	4 Mbytes/sec
Synchronous	5 Mbytes/sec
Fast	10 Mbytes/sec
Fast-20 (Ultra)	20 Mbytes/sec

The SCSI committees are working on fast-40 and fast-80 specifications.

SCSI Speeds

The SCSI bus supports four different speeds (the rate at which bytes are transferred along the cables). All SCSI devices support the async speed. Faster speed are negotiated with the device by the system device driver, and can usually be determined with the `prtconf` command (discussed later in this module).

The same bus may have devices operating at different speeds; there is no problem in doing this. Bus arbitration (the process of determining which device may use the bus) always occurs at async rate.

However, it may not be a good idea to put slow devices on the bus since they can slow the response of faster devices on the bus.

Fast-20 (commonly known as UltraSCSI) is the first of several high-speed extensions being worked on by the American National Standards Institute (ANSI) SCSI committee. Fast-40 and Fast-80 are in development.

Keep in mind that the SCSI speeds given in the table in the previous overhead are *maximums*. Like a speed limit, these are speeds that are not to be exceeded. They do not require the device to operate at that speed; it can operate at a slower rate.

You can limit the speed at which the bus will transfer, and the use of some SCSI capabilities, such as tagged queueing, by setting the *scsi_options* parameter in the `/etc/system` file. Normally, this would not be necessary, but for some devices it may be required. It can also be set (through the OpenBoot™ PROM [OBP]) on an individual SCSI controller basis.



Sun Educational Services

SCSI Widths

Name	Width	Cable
Narrow	1 byte	50 conductor cable
Wide	2 bytes in parallel	68 conductor cable
Quad	4 bytes in parallel	2 cables, 104 conductors

- Quad SCSI is not usually available.
- Speed and width together determine the bandwidth.
 - Multiply speed by width

SCSI Widths

The width of a SCSI bus specifies how many bytes may be transferred in parallel down the bus. Multiplied by the speed, this gives the bandwidth of the bus.

Just like speed, width is negotiated by the driver and device at boot time, and can be mixed on the same bus. Again, `prtconf` will provide the current setting for a device.

Each width uses a different cable. While quad devices are essentially nonexistent, it is possible to use an adapter cable to mix narrow and wide devices on a bus, regardless of whether the host adapter is narrow or wide.

Keep in mind that a wide bus, once it has been connected to narrow devices, cannot be made wide again.



SCSI Lengths

Name	Limit	Symbol
Single-ended	20 feet, 6m	
Differential	Sync – 66 feet, 20m Fast – 82 feet, 25m	

- Subtract 18 inches or .5 meters for each device on the bus

Do not mix differential and single-ended devices on the same bus. You could damage the equipment.

SCSI Lengths

Unlike speed and width, the length capability of a SCSI device is manufactured in, and cannot be changed or negotiated.

The electrical characteristics of single-ended and differential buses are different, so never mix device types on a bus. If you are unsure of the device type or bus type, look for the identifying logo on the device or interface board. While there are also specific terminators, cables will work in either environment. As long as you use an active terminator (with the light-emitting diode [LED]), the terminator has no influence on total bus length. (A passive terminator can greatly limit total length.)

The SCSI bus is limited, in either case, to the distance that the signals can travel before enough voltage is lost to make distinguishing a zero from a one unreliable. The lengths given in the above table are maximums; every device on the bus and every cable join (butt) takes an additional 18 inches (0.5 meter) of length of the total.

Remember that a shorter cable will always provide better performance than a long one.



Sun Educational Services

SCSI Properties Summary

Speed	Width	Length
Async	Narrow	Single-ended
Sync	Wide	Differential
Fast	Quad	
Fast-20		

- Each device runs with one combination.
 - Choose one from each column
- Multiple combinations are supported on one bus.
- Do not mix single-ended and differential devices.

SCSI Properties Summary

Every SCSI device on a bus operates with a set of characteristics designed into it and negotiated with the driver. While there are common mixtures, such as fast-wide, uncommon mixtures are acceptable.

Usually no message is given (nor is information made available) by the system describing the mode a device is operating in.

Since you might have devices of different capabilities on the same bus, it is not uncommon to find performance problems resulting from this. A slower device may prevent a faster device from performing an operation, sometimes for a significant period of time. This will be discussed later in this module.



SCSI Interface Addressing

- A SCSI *target* is a device controller, not a device.
- There are 8 unique targets on a narrow bus, 16 on a wide.
- Each target supports eight luns, or devices.
- Embedded SCSI has one lun per target.
- The tXdX scheme identifies these targets (+) and luns.
- SCSI commands are issued by initiators.
 - The most common initiator is the host adapter.
 - Host adapters do not have luns.
- There are 57 (narrow) or 121 (wide) positions on the bus.

SCSI Interface Addressing

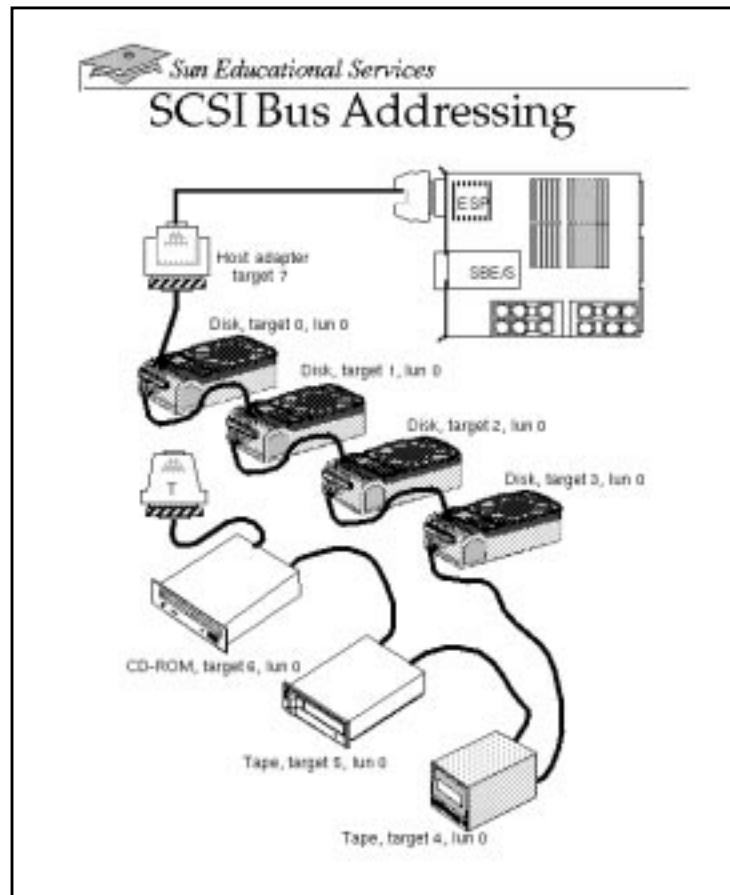
On a narrow SCSI bus, there are 57 possible devices on a wide bus 121. It is commonly thought that there are only 8, which is an understandable misconception.

A SCSI *target* is not a device, it is a device controller. It specifies an attachment point on the bus where as many as eight devices (called luns or logical unit numbers) may be attached. This is reflected in the tXdX notation that is used for SCSI devices, showing the target and device lun addresses.

Embedded SCSI devices support only one lun per target. Most SCSI devices available today are embedded, causing the confusion about the number of devices supported on the bus.

A narrow bus supports 8 targets, a wide bus 16. (This is due to the use of cable data lines to arbitrate bus usage.) The host adapter has only one lun.

SCSI commands are issued by an *initiator*, a device on the bus that can initiate commands. The most common initiator is the host adapter, the system's interface to the SCSI bus. Although it is rare, the standard does allow any device to be an initiator. While host adapter and initiator are used synonymously, they technically are different.



SCSI Bus Addressing

The overhead image shows a SCSI bus configured with embedded devices. The devices can be attached in any order, although more important devices should be attached closer to the host adapter if possible.

The SCSI terminator does not take a bus target position. It is invisible to the host adapter and devices, and is used to control the electrical characteristics of the bus.



SCSI Target Priorities

- The SCSI bus needs an *arbitration* scheme.
 - This determines which device goes first.
 - Higher priority devices get better service.
 - On a narrow bus, the priorities run 7–0.
 - On a wide bus they run 7–0 then 15–8.
 - The host adapter is usually 7.
 - Multi-initiator SCSI allows multiple initiators.
 - The host adapters are numbered 7 and 6.

SCSI Target Priorities

Most networks do not have a priority mechanism; whichever device needs to transmit can do so if the network medium is free.

This is not the case on a SCSI bus. Each target has a priority. When a device on a target wants to transmit, and the bus is free, an *arbitration* process is performed. Lasting about 1 millisecond, the arbitration phase requests all devices that need to transmit to identify themselves, and then grants the use of the bus to the highest priority device requesting it. Obviously, higher priority devices get better service.

The priority scheme is not obvious. On a narrow bus, priorities run from 7 (highest) to 0 (lowest). On a wide bus, for compatibility reasons, priorities run 7–0 then 15–8. The host adapter is usually 7, the highest priority target.

With multi-initiator SCSI, the initiators should be the highest priority devices on the bus.



Fibre Channel

- Is a reliable, scalable, gigabit interconnect technology
- Combines the best of "channels" and "networks" into a single I/O interface
- Operates on both fiber optic cable and copper wire
- Supports a variety of popular protocols:
 - SCSI
 - IP
 - ATM
 - IEEE 802.5 - Token Ring

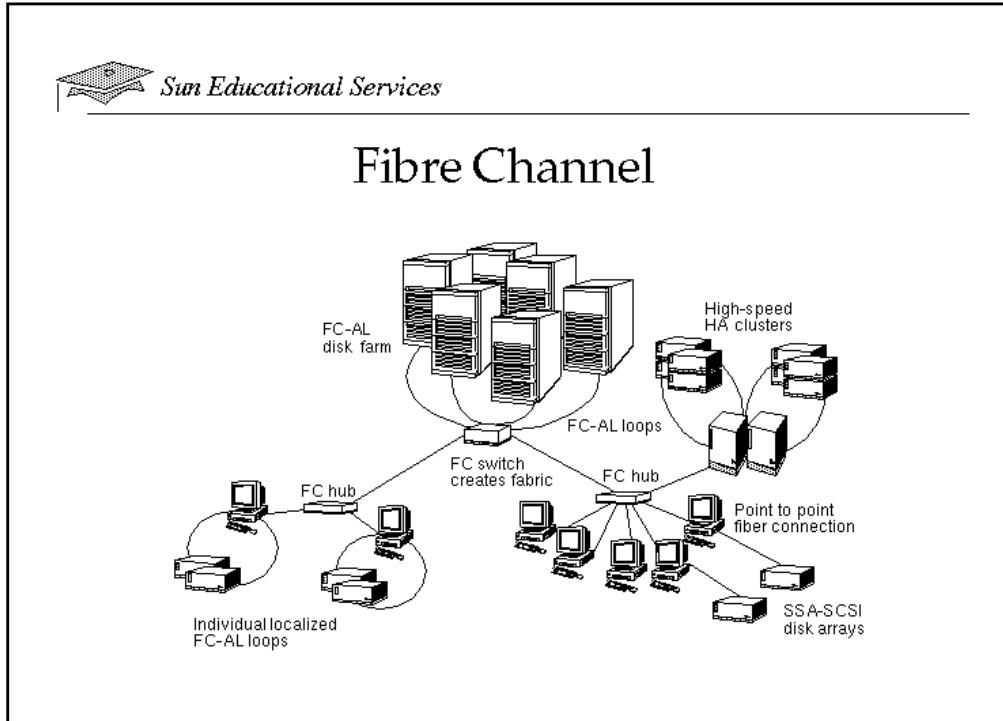
Fibre Channel

Fibre Channel (FC) is a reliable, scalable, gigabit interconnect technology allowing concurrent communications among workstations, mainframes, servers, data storage systems, and other peripherals.

Fibre Channel attempts to combine the best of "channels" and "networks" into a single I/O interface. The network features provide the required connectivity, distance, and protocol multiplexing. The channel features provide simplicity and performance.

Fibre Channel operates on both fiber cable and copper wire and can be used for more than just disk I/O. The Fibre Channel specification supports high-speed system and network interconnects using a wide variety of popular protocols, including:

- SCSI – Small computer system interface
- IP – Internet Protocol
- ATM – Adaptation layer for computer data (AAL5)
- IEEE 802.2 – Token Ring



Fibre Channel has been adopted by the major computer systems and storage manufacturers as the next technology for enterprise storage. It eliminates distance, bandwidth, scalability, and reliability issues of SCSI.

Fibre Channel Arbitrated Loop (FC-AL) was developed specifically to meet the needs of storage interconnects. Employing a simple loop topology, FC-AL can support both simple configurations and sophisticated arrangements of hubs, switches, servers, and storage systems. With the ability to support up to 126 FC-AL nodes (device objects) on a single loop, cost and implementation complexity are greatly reduced.

The FC-AL development effort is part of the ANSI/ISO (American National Standards Institute/International Standards Organization) accredited SCSI-3 standard, helping to avoid the creation of non-conforming, incompatible implementations. Virtually all major system vendors are implementing FC-AL, as are all major disk drive and storage system vendors.

Gigabit Fibre Channel's maximum data rate is 100 Mbytes/sec per loop, after accounting for overhead, with up to 400 Mbytes/sec envisioned for the future. This is faster than any Ultra-SCSI-3, serial storage architecture, or P1394 (Firewire).

In a Fibre Channel network, legacy SCSI storage systems are interfaced using a Fibre Channel to SCSI bridge. IP is used over FC for server to server and client to server communications.



Disk I/O Time Components

- Access to the I/O bus
- Other queued disk requests
- Bus transfer time
- Seek time (to find correct track in cylinder)
- Rotation time (to find correct sector in track)
- ITR transfer time
- Reconnection time
- Interrupt time

Disk I/O Time Components

A SCSI I/O operation goes through many steps. To determine where bottlenecks are, or to decide what must be done to improve response time, an understanding of these steps is important.

Access to the I/O Bus

An I/O operation cannot begin until the request has been transferred to the device. The request cannot be transferred until the initiator can access the bus. As long as the bus is busy, the initiator must wait to start the request. During this period, the request is queued by the device driver.

A queued request must also wait for any other requests ahead of it. A large number of queued requests suggests that the device or bus is overloaded. The queue can be reduced by several techniques: use of tagged queueing, a faster bus, or moving slow devices to a different bus.

Queue information is reported by the **sar -d** report and the **iostat -x** report.

```
# iostat -x
                                extended device statistics
device    r/s    w/s    kr/s    kw/s    wait   activ  svc_t   %w   %b
fd0      0.0    0.0    0.0     0.0    0.0    0.0    0.0    0     0
sd0     18.5   2.8   180.1  122.6   0.0    1.1    52.8   0    25
sd6      0.0    0.0    0.0     0.0    0.0    0.0    0.0    0     0
nfs1      0.0    0.0    0.0     0.0    0.0    0.0    0.0    0     0
nfs2      0.4    0.5    6.4     9.3    0.0    0.0    22.7   0     2
nfs3      2.0    0.0   43.3     0.0    0.0    0.0    7.8    0     1
#
```

The fields in this output are:

- **wait** – The average number of transactions waiting for service (queue length). In the device driver, a read or write command is issued to the device driver and sits in the wait queue until the SCSI bus and disk are both ready.
- **%w** – Percentage of time the queue is not empty; percentage of time transactions wait for service.

```
# sar -d 1
SunOS grouper 5.6 Generic_105181-03 sun4u      05/25/99

08:58:48  device        %busy    avque    r+w/s    blks/s    await    avserv
08:58:50  fd0          0        0.0      0        0        0.0      0.0
           nfs1          0        0.0      0        0        0.0      0.0
           nfs2          0        0.0      0        7        0.5      15.5
           nfs3          0        0.0      0        18       0.0      7.6
           sd0          6        0.1      4        97       0.0      32.1
           sd0,a        2        0.1      1        21       0.0      46.4
           sd0,b        4        0.1      3        76       0.0      26.7
#
```

In this output, one field is:

- **await** – Average time, in milliseconds, that transfer requests wait idly in the queue. (This is measured only when the queue is occupied.)

Bus Transfer Time

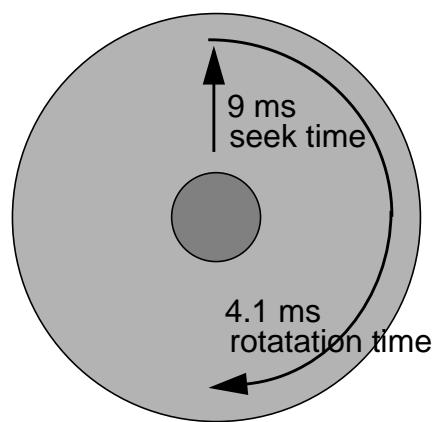
This includes several components: arbitration time, where the determination is made of which device can use the bus (the host adapter always wins), and the time required to transfer the command to the device over the bus. This takes approximately 1.5 milliseconds (MS). In the case of a write, this time also includes the data transfer time.

Seek Time

Seek time is the amount of time required by the disk drive to position the arm to the proper disk cylinder. It is usually the most costly part of the disk I/O operation. Seek times quoted by disk manufacturers are average seek times, or the time that the drive takes to seek across 1/2 of the (physical) cylinders. With tagged queueing active, this time is not representative of the actual cost of a seek.

Rotation Time

Rotation time, or latency, is the amount of time that it takes for the disk to spin under the read/write heads. It is usually quoted as 1/2 of the time a complete rotation takes. It is not added into the seek time when a manufacturer quotes disk speeds, although a figure for it is provided in RPM (revolutions per minute).



ITR Transfer Time

ITR time (internal throughput rate) is the amount of time required to transfer the data between the device's cache and the actual device media. It is usually much slower than the bus speed. ITRs are not always available for devices, although they may be found in the manufacturer's detailed specifications. ITRs are usually between 5 and 10 Mbytes per second. ITRs are the limiting factor on performance for sequential disk access. For random access patterns, I/O time is dominated by head seek and disk rotation time.

Reconnection Time

Once the data has been transferred to or from cache, the request has to be completed with the host adapter. An arbitration and request status transfer must be performed, identical to that done at the start of the request. The only difference is that the arbitration is done at the device's priority, not that of the host adapter.

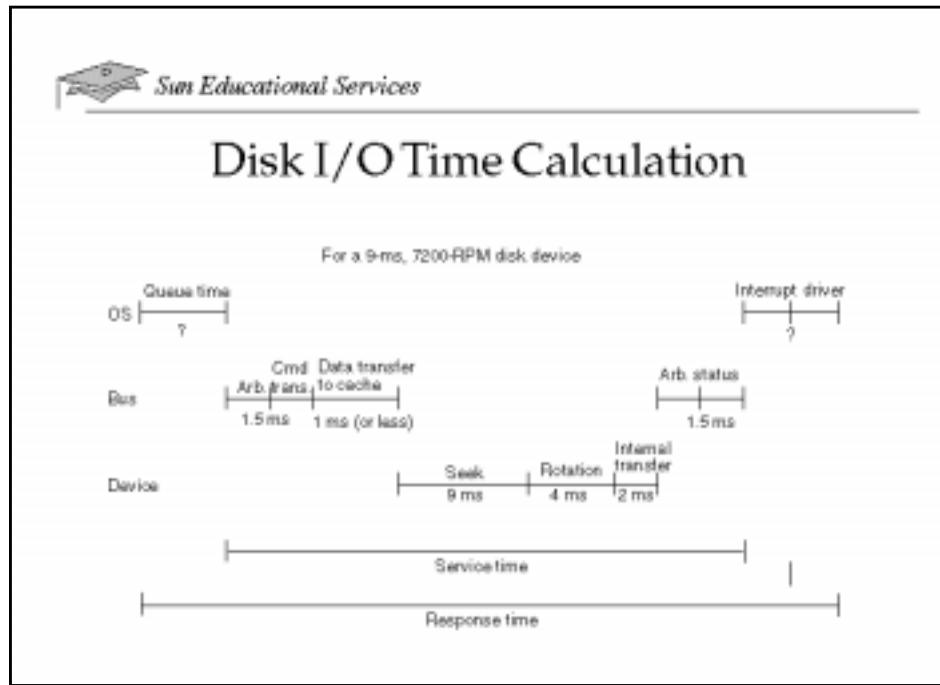
If the request was a read, the data is transferred to the system as part of this operation.

Timing is similar to host adapter arbitration, about 1.5 ms, plus any data transfer time.

Interrupt Time

This is the amount of time that it takes for the completion interrupt for the device to be handled. It includes interrupt queueing time (if higher priority interrupts are waiting), time in the interrupt handler, and time in the device driver interrupt handler.

It is very difficult to measure; however, if a high interrupt rate on CPUs on the system board to which the device is attached is observed, and no other cause for delay can be seen, this may be part of the problem. Move the host adapter or higher interrupt priority devices to another system board if necessary.



Disk I/O Time Calculation

This overhead image shows the calculation of the total cost of an I/O operation.

Remember, response time is the total, elapsed time for an operation to complete. Service time is the actual request processing time, ignoring any queueing delays.

Some of the costs are fixed, such as arbitration and command/status transfer; some are device specific, such as ITR, seek time, and latency; and some are unpredictable, such as queueing time and interrupt time.

The cost of an operation on a particular device can be calculated, given a knowledge of the device characteristics and average data transfer size.

The following tables can help in the cost of operation calculations.

Bus Data Transfer Timing

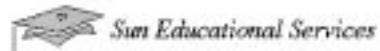
Table 8-1 Bus Data Transfer Timing

Type	Speed	1 Sector	8 Kbytes
Asynchronous/narrow	4 Mbytes/sec	.128 ms	2.05 ms
Synchronous/narrow	5 Mbytes/sec	.102 ms	1.63 ms
Fast/narrow	10 Mbytes/sec	.051 ms	0.82 ms
Fast/wide	20 Mbytes/sec	.025 ms	0.41 ms
Ultra/wide	40 Mbytes/sec	.013 ms	0.21 ms

Latency Timing

Table 8-2 Latency Timing Calculation

Revolutions per Minute (RPM)	Revolutions per Second (RPS)	Milliseconds per Revolution	Milliseconds per 1/2 Revolution
3600	60	16.7	8.33
5400	90	11.1	5.55
7200	120	8.33	4.17
10,000	167	5.98	2.99



Disk Drive Features

- Multiple zone recording
- Drive caching and fast writes
- Mode pages
- Tagged queueing
- Properties from `prtconf` and `scsiinfo`

Disk Drive Features

Disk drives have become complex devices. Drives today contain upwards of 8 Mbytes of DRAM memory and custom logic with Intel 80286 microprocessor cores. They perform a lot of functions besides basic data retrieval.

These features are discussed over the next few pages.



Sun Educational Services

Multiple Zone Recording

Zone table

Sector	Density	Cylinder
0	75	0
350,000	70	88
685,000	65	176
	60	
	55	
	...	



Multiple Zone Recording

Disk drives are presented to the OS as having a particular geometry. For example, the device may report 2200 cylinders, 16 heads, and 64 sectors per track. A simple examination of the device shows that this cannot be the real geometry: there are not 16 recording surfaces in a drive that is only 1 inch high.

Manufacturers are very creative when it comes to increasing the capacity of their devices. The marketing department did their job: a drive listed at 1.02 billion bytes is a 1.0 Gbyte drive, since a Gbyte has been defined as 1 billion bytes, not 1024 Mbytes. This, however, is only good for a 2 percent increase.

The technical approach used to increase the capacity is called multiple zone recording (MZR), also known as zone bit recording (ZBR).

This technique takes advantage of the ability of a drive to write very densely on inner, smaller cylinders. While it would be possible to write this densely on outer, larger cylinders, this would give the drive a varying geometry, making it very difficult for the OS to work with.

With the high-capability microprocessor in the drive, this problem can be overcome. A nominal geometry is reported to the OS, which the OS uses in formulating requests to the drive. The final request is sent as an absolute sector number from the beginning of the drive.

The MZR drive is divided into zones, usually 12 to 14, each of which have a constant density. As the zones move inward, the number of sectors on each track decreases. The drive contains a table of which sectors are in each zone. There might be 60 sectors in an inner zone and 120 or more in an outer zone.

The drive then takes the requested sector number, translates it using the zone table, and determines the actual physical location on the drive to be accessed. The varying, true geometry of the drive is hidden from the OS.

This puts denser cylinders on the outside of the drive: more data can be accessed without moving the arm. This difference is noticeable. When accessing data sequentially, the performance difference from the outermost to the innermost cylinders can be as much as 33 percent. MZR also improves the ITR for this area of the disk.

When partitioning a ZBR disk, remember, slice 0 will be faster than slice 7.



Drive Caching and Fast Writes

- Disk drives today have from .5 Mbyte to 8+ Mbytes of cache.
- Cache is used with tagged queueing to manage requests.
- It also buffers the drive's ITR to bus speed.
- Fast writes signal completion when the data reaches the cache.
 - You should have the drive on a UPS to use fast writes.
- It can be set in arrays or for individual drives.

Drive Caching and Fast Writes

Disk drives, and most SCSI devices, today contain significant amounts of data cache. When an I/O request is made, data is transferred in and out of cache on the device before the actual device recording mechanism is used. This allows the device to transfer data at the speed of the bus rather than at its more limited ITR.

Individual SCSI devices today have from .5 to 8 Mbytes or more of cache. Some storage arrays have multiple Gbytes. In association with the tagged queueing feature (discussed on page -29), multiple requests may be active concurrently on the device.

Fast Writes

Fast writes allow the device to signal completion of a write operation when data has been received in the cache, before it is written to the media. (The disk drive's cache becomes write back instead of write through.)

While providing significant speed increases (as discussed later in this module), it is very unsafe unless used with a UPS (uninterruptable power supply). Loss of power could cause the data to be lost before it is written to the media, while the OS has already been told that the write was successful.

Fast writes can be set on an individual drive basis (although an OS interface to do so is not usually available), or more commonly, in storage arrays.



Tagged Queueing

- Adds a special ID field to the device command
- Allows the drive to handle multiple simultaneous requests
- Sometimes executes requests out of order
- Takes advantage of ordered requests to limit arm motion
- Is negotiated with the device by the driver
 - Not all devices support it
- Can provide noticeable performance improvements

Tagged Queueing

Tagged queueing is a mechanism that allows the device to support multiple concurrent I/O operations. Its use is negotiated between the device and the driver. Not all devices support it. (The `prtconf` and `scsiinfo` commands can be used to determine if a specific device is using it.) Solaris 2.4 and higher operating systems have tagged queueing enabled by default, at a depth of 32.

In addition to the regular SCSI command, an extra ID field is added which identifies the request to the OS device driver. When the request completes, the tag is returned with a request status that identifies to the driver which command has completed.

In and of itself, multiple commands do not provide a significant performance advantage. The advantage comes from the ability of the device to optimize its operation. This is done through ordered seeks.

Ordered Seek

With tagged queueing, multiple requests are sent to the disk for execution. If executed in the order in which they were received, the total motion of the disk arm, the seeks, would be the sum of all distances between the requests.

With *ordered seeks*, sometimes called *elevator seeks*, the arm makes passes back and forth over the drive. As the location for a request is reached, the request is executed. While the requests are fulfilled out of order, the arm motion is greatly reduced. The average execution time for a request is significantly less than it would be otherwise.

As the number of simultaneous requests to the disk increases, average request response time will improve (to a certain point), since the average arm movement for a request will decrease. The device operates more efficiently under load.

To avoid long delays for requests at the opposite end of the drive, requests that arrive after a pass has started are queued for the return pass.

Response time improvements up to 2/3 have been measured.



Mode Pages

- Are provided for all SCSI devices
 - Identified by number
- Describe device properties and configuration
- Have several standard, required pages
- Have additional device-specific pages
- Can include vendor-specific pages
- Can be viewed using `format -e`
- Have header files in `/usr/include/sys/scsi`

Mode Pages

You may have noticed that it is no longer necessary to provide `format.dat` entries for your disk drives. This is because SCSI devices provide a descriptive interface called *mode pages*.

Mode pages are blocks of data that are returned by a SCSI device in response to the appropriate request. The SCSI standard defines several specific mode pages that must be supported by every SCSI device, and some additional ones for certain device types. Device manufacturers usually add their own vendor-specific mode pages to the device. Specific mode pages are identified by number.

By interrogating the mode pages, many of the device characteristics can be determined by the driver or OS. In general, they are not visible to the user, although they can be viewed by using the expert mode of the `format` command (the `-e` option) and then invoking the `scsi` command.

Mode pages can also be written to, to change device behavior.



Device Properties From prtconf

- `prtconf -v` reports some device information.

```
name <target1-TQ> length <4>
      value <0x00000001>.
name <target1-wide> length <4>
      value <0x00000001>.
name <target1-sync-speed> length <4>
      value <0x00004e20>.
```

- What is reported depends on the device.
- Device properties are listed under the controller.
- This device is using a fast-wide SCSI bus and tagged queueing.

Device Properties From prtconf

The `prtconf` command can be used to format the OpenBoot PROM device tree. (Explanation of the OBP device tree is beyond the scope of this course.) The OBP contains a node for each device that the OBP finds, or is attached by a peripheral bus. By using the `prtconf -v` command, the attributes of these device nodes can be seen.

For SCSI devices, this includes properties such as bus speed (`targetX-sync-speed`, a hexadecimal number in Kbytes per second), support for wide buses (the `targetX-wide` property is present), and tagged queueing (the `targetX-TQ` property is present).

This information can also be retrieved from the device mode pages.

Table 8-3 shows the speeds reported by `prtconf` and the translated bus speed.

Table 8-3 `prtconf` Bus Speeds

<code>prtconf</code> Speed	Bus Speed
9c40	40 Mbytes/sec
4e20	20 Mbytes/sec
2710	10 Mbytes/sec
1388	5 Mbytes/sec



Device Properties From `scsiinfo`

- `scsiinfo` can read mode page properties and device characteristics.
- It is not part of SunOS.
 - Available from <ftp://ftp.cs.toronto.edu/pub/jdd/scsiinfo/>
 - Provided in source code
- `scsiinfo` is the easiest way to see how your SCSI devices are operating

Device Properties From `scsiinfo`

The `scsiinfo` command can provide the same type of SCSI configuration information as `prtconf`, however, it is specifically designed for use with SCSI devices, and is more detailed and easier to read.

The `scsiinfo` command is not provided with the SunOS operating system, but can be downloaded from the World Wide Web. The URL is <ftp://ftp.cs.toronto.edu/pub/jdd/scsiinfo/>.

Only Sun workstations with esp, isp, fas, ptisp, and glm SCSI controllers (sun4c/4m/4e/4d/4u) running SunOS 4.1 or later are supported. The isp, fas and glm controllers are only supported under the SunOS 5.x environment.

Examples of output from **scsiinfo** are as follows:

```
# ./scsiinfo -o
esp0: sd0,0 tgt 0 lun 0:
    Synchronous(10.0MB/sec) Clean CanReconnect
esp0: sd2,0 tgt 2 lun 0:
    Synchronous(10.0MB/sec) Clean CanReconnect
esp0: sd3,0 tgt 3 lun 0:
    Synchronous(10.0MB/sec) Clean CanReconnect
esp0: sd6,0 tgt 6 lun 0:
    Synchronous(4.445MB/sec) Clean CanReconnect
esp0: st4,0 tgt 4 lun 0:
    Synchronous(5.715MB/sec) Noisy CanReconnect
esp0: st5,0 tgt 5 lun 0:
    Asynchronous Noisy CannotReconnect
#
# ./scsiinfo -r /dev/rdsk/c0t3d0s0
Vendor:          SEAGATE
Model:           ST31200W SUN1.05
Revision:        8724
Serial:          00440781
Device Type:     Disk
Data Transfer Width(s): 16, 8
Supported Features: SYNC_SCSI TAG_QUEUE
Formatted Capacity: 2061108 sectors (1.03 GB)
Sector size:      512 bytes
Physical Cylinders: 2700
Heads:            9
Sectors per track (Avg.): 85
Tracks per zone:  9
Alternate Sectors per zone: 9
Alternate Tracks per volume: 18
Rotational speed: 5411 rpm
Cache:            Read-Only
#
```



Sun Educational Services

I/O Performance Planning

Disk	Sequential Reads Mbytes/Second	Random Reads Operations/Second
Elite 3	4.5	60
Elite 9	5.5	60
SSA RSM drive	5.5	60
Baracuda 4	6.5	75
Baracuda 9	7.5	75

Controller	Sequential Reads Mbytes/Second	Random Reads Operations/Second
ISP	14	1700
SOC	18	2400

I/O Performance Planning

The characteristics of the I/O operations can have a dramatic impact on the throughput that the bus can sustain. Sequential operations run at the drive's ITR, limiting the bus performance to that of the drive. Random operations run at bus speed, allowing many more operations.

When you are planning a disk configuration, you must design for the type of load the system will create.

Decision Support Example (Sequential I/O)

For a decision-support or data-warehousing application, 700 Mbytes per second throughput is required. ISP (intelligent SCSI processor) controllers will be used. Use the following calculations:

$$700 \text{ Mbytes per second} \div 14 \text{ Mbytes per controller} = 50 \text{ ISP controllers}$$

$$700 \text{ Mbytes per second} \div 5.5 \text{ Mbytes per RSM drive} = 128 \text{ drives}$$

The designed configuration would need 50 ISP controllers, with only two or three devices per controller.

Transaction Processing Example (Random I/O)

For a transaction-processing application, 30,000 I/O operations per second throughput is required. SOC controllers (for SPARCstorage Arrays [SSA]) will be used. Use the following calculations:

$$30,000 \text{ I/O operations per second} \div 2400 \text{ operations per SOC controller} = 18 \text{ SOC controllers}$$

$$30,000 \text{ I/O operations per second} \div 75 \text{ operations per SSA RSM drive} = 400 \text{ RSM drives}$$

The designed configuration would need 18 SOC controllers, with 23 devices per controller.



Tape Drive Operations

- Bus speed is a speed limit, not a mandatory speed.
- Tapes can seriously hurt your bus performance.
 - They transfer very slowly.
 - Other devices have to wait until the tape is done.
- Tapes should be on a separate bus from production disks.
- Backups can seriously degrade bus performance.

Tape Drive Operations

Remember that bus speed is a maximum, not a requirement. Tape drives, even though put on a fast bus, usually cannot transfer faster than their tape motion allows.

If a tape drive transfers slowly, say at synchronous speed, it will hold the bus for 1 ms for an 8-Kbyte transfer, even if it is on a fast/wide bus. Since the tape must be a high priority device (missed requests are expensive to restart), its effect on other devices on the bus can be dramatic.

This kind of effect is known as SCSI starvation, where a higher priority device, whether very active or just slow, prevents lower priority devices from receiving adequate service.

The best solution is to move the slow device(s) to another bus, or try to split the load of highly active devices.



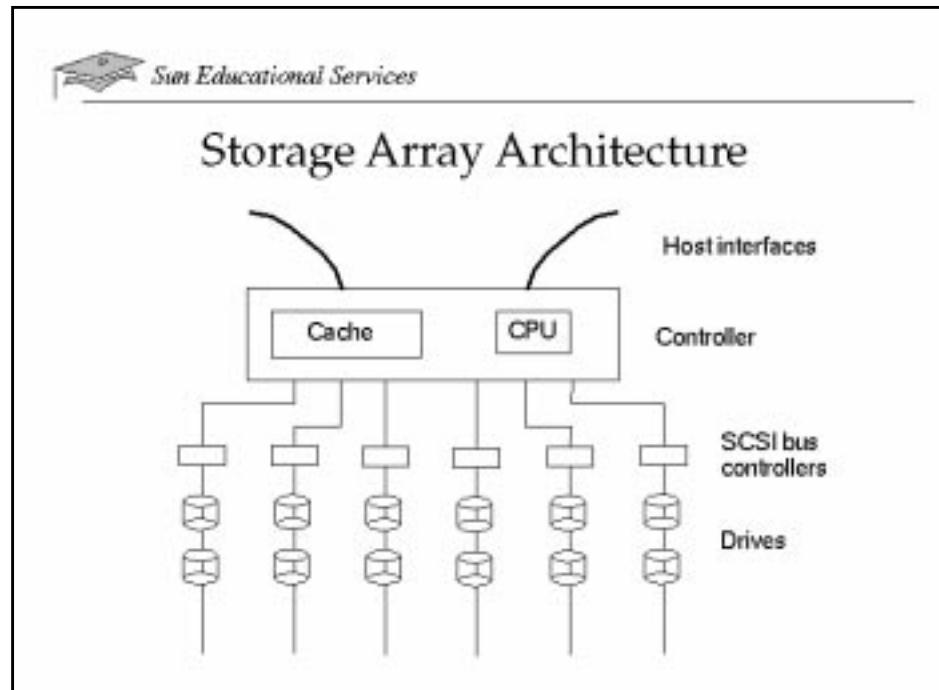
Storage Arrays and JBODs

- JBOD (A5000, MultiPack)
 - Looks like a bus in a box
 - Has no controller and no special control operations
 - Might have special power and reliability support
- Storage arrays (SPARCstorage™ Array, A7000)
 - Has a controller and cache in front of multiple buses
 - Must be slower than plain disks if the cache fills
 - Provides many additional features

Storage Arrays and JBODs

A JBOD (just a bunch of disks) is a storage device that provides a compact package for a number of disk drives, usually with a common power source. It does not provide any other functions such as caching or using RAID. The devices are individually visible on the bus. Examples are the Sun Multi-Pack and the Sun StorEdge™ A5000.

A storage array provides a compact package with additional management for the devices. It may also provide caching, RAID, transparent failed-drive replacement, or special diagnostic capabilities. Drives may or may not be individually visible outside of the array. Examples are the SPARCstorage™ Array, the A3000, and the A7000.



Storage Array Architecture

Conceptually, a storage array is quite simple. It is a box of SCSI buses.

The array provides a single interface (the *front end*) into a series of SCSI buses accessing the array drives (the *back end*). The front end is configured as any individual storage device and assigned an appropriate target number. (Customarily the array will be the only device on the bus due to its high I/O rate.)

The back end is tuned as a series of SCSI buses. Use of the drives is planned just as if they were drives configured on buses that are not in an array. Target priorities apply as in a regular SCSI bus.

Note that a storage array, depending on its caching ability, may be slower than the equivalent number of individual drives, especially on reads, due to controller overhead. Also, devices reported by a storage array in the performance tools may not correspond to actual physical devices. Be sure you know what they represent physically.



RAID

- Stands for redundant array of independent disks
- Combines multiple disk drives into a single logical unit
- Overcomes inherent limitations of a single disk drive
- Is designed to enhance both reliability and performance
- In the original UC-Berkeley papers, is defined as six RAID levels
- Has different levels for different environments

RAID

As defined by the original University of California at Berkeley research papers, RAID (redundant array of independent disks) is a simple concept. Several disks are bound together to appear as a single, large storage device to the host system. The goal of RAID units in general is to overcome the inherent limitations of a single disk drive and to improve I/O performance and data storage reliability.

There are six levels of RAID, numbered from 0 through 5, designed to address either performance or reliability. Some RAID levels are fast, some are inexpensive, and some are safe. None of the RAID levels deliver more than two of the three. Today, RAID levels 2 through 4 are rarely, if ever, seen. The most common implementations provide, through hardware or software (or both), RAID levels 0, 1, or 5.

As with any technology, there are cases where the use of certain RAID levels is appropriate and cases where it is not. The incorrect use of a particular RAID level, especially RAID-5, can destroy the system's I/O performance.



RAID-0

- RAID-0 is striping (spreading data across multiple drives).
- RAID-0 offers no data protection or redundancy.
- It is effective in both sequential access and random access environments.
- You should tune the stripe size according to the data access pattern.
- You must keep in mind SCSI overhead when selecting stripe size.
- You can distribute member disks across multiple controllers.

RAID-0

RAID-0 provides significant performance benefits, but no redundancy benefits. Instead of concentrating the data on one physical volume, RAID-0 implementations *stripe* the data across multiple volumes. This allows the single file or partition to be accessed through multiple drives.

Striping improves sequential disk access by permitting multiple member disks to operate in parallel. Striping also improves random access I/O performance though not directly. Striping does not benefit individual random access requests. But striping can dramatically improve overall disk I/O throughput in environments dominated by random access by reducing disk utilization and disk queues.

When more I/O requests are submitted than the disk can handle, a queue forms and requests must wait their turn. Striping spreads the data evenly across all of the volume's member disks in order to provide a perfect mechanism for avoiding long queues.

Tuning the stripe size is very important. If the stripe size is too large, many I/O operations will fit in a single stripe and be restricted to a single drive. If the stripe is too small, many physical operations will take place for each logical operation, saturating the bus controller.

Performance in random access environments is maximized by including as many member disks as possible. Disk utilization, disk response latency, and disk queues are minimized when the number of disks is maximized. This optimal result is accomplished with a relatively large stripe size, such as 64 Kbytes or 128 Kbytes.

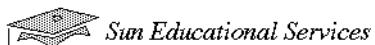
Performance in sequential access environments is maximized when the size of the I/O request is equal to the strip size. Match the stripe size to a multiple of the system's page size, or a common application's I/O parameter. Oracle, for example, blocks all I/O into 8 KByte operations; a four-drive RAID-0 group with a 2-Kbyte stripe size would balance each Oracle read or write across all four drives in the RAID group.

Keep in mind, however, that each request to a member disk requires the generation of a SCSI command packet with all of its associated overhead. For small requests, the effort of transferring the data is trivial compared to the amount of work required to set up and process the I/O. Typically the break-even point is a strip size of 2 Kbytes to 4 Kbytes.

When you initiate many physical I/O operations for each logical I/O operation, you can easily saturate the controller or bus to which the drives are attached. You can avoid this problem by distributing the drives across multiple controllers. An even better solution would be to put each drive on a private controller.

None of the drives on a single SCSI bus should be part of the same RAID group. Adding more controllers and managing I/O distribution across those controllers is critical to achieving optimal performance from a RAID configuration.

The drawback to RAID-0 is that it is not redundant. The loss of any physical volume in the RAID-0 configuration usually means the entire contents of the configuration needs to be restored.



RAID-1

- RAID-1 is mirroring (copying data to multiple locations).
- RAID-1 provides data redundancy.
- Multiple controllers are necessary for performance.
- Read workload can be distributed for better performance.
- Writes are serviced in parallel.
 - Typical writes to mirrors take only about 15 to 20 percent longer than writes to individual disks.
- The main drawback is the cost of extra disks for mirrors.

RAID-1

RAID-1 is disk mirroring. Every time data is written to disk, one or more copies of the data are written to alternate locations. This provides data redundancy in the event of the loss of a disk device. When a drive fails, the system simply ignores it, reading and writing data to its remaining partner in the mirror. Properly configured, a RAID-1 drive group can survive the loss of an entire I/O bus and even a storage array unit.

A RAID-1 group running with one or more failed or missing drives is said to be running in degraded mode. Data availability is not interrupted, but the RAID group is now open to failure if one of the remaining partner drives should fail. The failed drive needs to be replaced quickly to avoid further problems.

When the failed drive is replaced, the data in the good drive must be copied to the new drive. This operation is known as synching the mirror, and can take some time, especially if the affected drives are large. Again, data access is uninterrupted, but the I/O operations needed to copy the data to the new drive steal bandwidth from user data access, possibly decreasing overall performance.

By its nature, RAID-1 can double the amount of physical I/O in your system, especially in a write-intensive environment. For this reason, the use of multiple controllers is an absolute necessity for effective RAID-1 configurations.

Some RAID-1 implementations provide the ability to read from any mirror, thus distributing the read workload for better performance.

A common misperception is that writes to a mirror are half the speed of writes to an individual disk. This is rarely the case since the two writes do not need to be serviced sequentially. Most RAID-1 implementations handle mirrors the same way they handle stripes; the writes are dispatched sequentially to both members, but the physical writes are serviced in parallel. As a result, typical writes to mirrors take only about 15 to 20 percent longer than writes to a single member.

The main drawback to RAID-1, besides the duplicate write overhead, is cost. All RAID-1 storage is duplicated, thus doubling the physical disk storage cost. Considering the costs of downtime, reconstruction of missing data, and extensive logging and backup for non-mirrored data, RAID-1 can easily pay for itself.



RAID-0+1

- RAID-0+1 is mirrored stripes.
- RAID-0+1 provides the performance of RAID-0 and the redundancy of RAID-1.
- It is often implemented in hardware using smart mirroring controllers.
- Sequential reads improve through striping.
- Random reads improve through the use of multiple disks.
- Writes are about 30 percent slower than RAID-0 writes.

RAID-0+1

Pure RAID-1 suffers from the same problems as pure RAID-0: data is written sequentially across the volumes, potentially making one drive busy while the others on that side of the mirror are idle. You can avoid this problem by striping within your mirrored volumes, just like you striped the volumes of your RAID-0 group.

Theoretically, RAID-1 mirrors physical devices on a one-for-one basis. In reality, you will want to build big logical devices using RAID-0 groups and then mirror them for redundancy. This configuration is known as RAID 0+1, since it combines the concatenation features of RAID-0 with the mirroring features of RAID-1.

Unlike RAID-0, RAID-1 and RAID 0+1 are often implemented in hardware using smart mirroring controllers. The controller manages multiple physical devices and presents a single logical device to the host system. The host system sends a single I/O request to the controller, which in turn creates appropriate requests to multiple devices in order to satisfy the operation.

Higher-end mirrored disk subsystems also offer hot-swap drives and some level of drive management tools. Hot-swap is an important feature, which enables you to remove and replace a failed drive without having to shut down the bus to which the drive is attached.

RAID-0 and RAID-1 exist at different ends of the RAID spectrum. RAID-0 offers large volumes with no redundancy or failure immunity at the lowest possible cost. RAID-1 provides complete data redundancy and robust failure immunity at a high cost. Both configurations can be tuned for performance by adding controllers and using striping to distribute the I/O load across as many drives as possible.

Sequential reads improve through striping and random reads improve through the use of many member disks. Writes typically approach the speed of RAID-0 stripes, but due to the overhead of writing two copies for mirroring, RAID-0+1 is about 30 percent slower.

RAID 0+1 takes the best of both configurations, providing large volumes, high reliability, and failure immunity. It does not, however, solve the price problem.



RAID-3

- RAID-3 is striping with a dedicated parity disk.
- Parity bits or bytes are computed from the data.
- The parity information is stored on a separate disk.
- Data can be reconstructed from the parity information.
- It performs well when servicing large quantities of single-stream sequential I/O.
- It does not perform well for random access I/O because of the physical organization of the RAID-3 stripe.
- Its reads perform as well as RAID-0 reads since parity data does not have to be accessed.

RAID-3

RAID-3 provides both the striping benefits of RAID-0 and the redundancy benefits of RAID-1 in less disk space than RAID-1. The idea behind RAID-3 is to keep one striped copy of the data, but with enough extra data (called *parity* data) to re-create the data in any stripe. This means that one physical volume could be lost, and the system would continue to operate by re-creating the lost data from the parity stripe (or by re-creating the lost parity from the data). The process is very similar to that of error checking and correction (ECC) used in main storage.

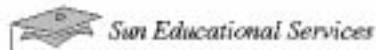
RAID-3 adds an additional disk to the member group. The additional disk is used to store the parity data. The parity disk contains corresponding blocks that are the computed parity for the data blocks residing on the non-parity members of the RAID group. Parity computation is quite fast. For a typical 8-Kbyte file system I/O, parity generation takes much less than a millisecond.

When servicing large quantities of single-stream sequential I/O, RAID-3 delivers efficiency and high performance at a relatively low cost. The problem with RAID-3 is that it does not work well for random access I/O, including most relational database systems and timesharing workloads.

Random access suffers under RAID-3 because of the physical organization of a RAID-3 stripe. Every write to a RAID-3 group involves accessing the parity disk. RAID-3 employs only one parity disk and it becomes the bottleneck for the entire group.

In most implementations, reads do not require access to the parity data unless there is an error reading from a member disk. Thus, from a performance standpoint, RAID-3 reads are equivalent to RAID-0 levels.

In practice, few vendors actually implement RAID-3 since RAID-5 addresses the shortcomings of RAID-3 while retaining its strengths.



RAID-5

- RAID-5 is also striping with parity.
- Parity information is striped across all member disks.
- It eliminates the single parity disk bottleneck of RAID-3.
- It is optimized for random access I/O patterns.
- Its writes use a two-phase commit protocol for data integrity.
- Non-volatile memory is frequently used to accelerate the write operations.

RAID-5

The primary weakness of RAID-3 is that writes overutilize the parity disk. RAID-5 overcomes this problem by distributing the parity across all of the member disks. All member disks contain some data and some parity. Whereas RAID-3 performs best when accessing large files sequentially, RAID-5 is optimized for random access I/O patterns.

A simple write operation results in two reads and two writes, a four-fold increase in the I/O load. These multiple writes for a single I/O operation introduce data integrity issues. To maintain data integrity, most RAID software or firmware uses a commit protocol similar to that used by database systems: (1) Member disks are read in parallel for parity computation; (2) the new parity block is computed; (3) the modified data and new parity are written to a log; (4) the modified data and parity are written to the member disk in parallel; and (5) the parts of the log associated with the write operation are removed.

Many sites use RAID-5 in preference to RAID-1 because it does not require the hardware cost of fully duplicated disks. However, RAID-5 is not suited for certain types of applications. The cost of a simple write operation involves a read to obtain parity data, the parity computation, two writes to manage the two-phase commit log, one write for the parity disk, and one or two writes to the data disks. All told, this represents a six-fold increase in actual I/O load and typically about a 60 percent degradation in actual I/O speed.

Non-volatile memory is frequently used to accelerate the write operations. Buffering the modified data and parity blocks is faster than writing them to physical disks. Using non-volatile memory also allows write cancellation.



Tuning the I/O Subsystem

- Remember, "The fastest I/O you can do is the one you do not do."
 - Remember, main memory problems cause I/O problems
 - Start by tuning main memory and UFS caches
 - Spread activity across boards, buses, and devices
 - Ensure devices are operating as they are supposed to
 - Calculate "good" I/O times
 - Eliminate hot spots
 - Use target and device priorities to place data

Tuning the I/O Subsystem

The best way to tune the I/O subsystem is to cut the number of I/O operations that are done. This can be done in several ways: requesting larger blocks (transfer time is a small part of the overall cost of an I/O operation), tuning main memory properly, tuning the UFS caches appropriately, and using write cancellation.

Remember, the I/O subsystem pays the price for memory problems, since problems with main memory (improper cache management) generate more I/O operations due to the need to move data in and out of storage. Do not spend a lot of time tuning the I/O subsystem if you have significant memory performance problems.

Spread the I/O activity across as many devices and buses as possible to try to avoid hotspots or overloaded components. `iostat` can be used to measure how well disk loads are spread across each disk on a server system. `sar -d` can be used to capture information about partition usage.

Determine whether your system is performing properly. You may spend time trying to tune poor I/O performance only to find that it is performing as well as it can. Calculate what your response times should be to see how close you are to achieving them.

Determine what your critical data is and place it appropriately in your I/O subsystem. Take into account bus speeds; SCSI targets; and device speeds and capabilities such as tagged queueing, tape drive placement, and data placement on the drive itself. Put your important data where you can get at it more quickly.

Remember, as you eliminate CPU and memory bottlenecks, you may see new I/O bottlenecks.



Sun Educational Services

Available Tuning Data

- iostat
- sar
- scsiinfo and prtconf

Available Tuning Data

There are a number of sources for data on the performance and configuration of your I/O subsystem.



Sun Educational Services

Available I/O Statistics

Activity	Source	iostat (unit)	sar (unit)
Simple time disk was active	(calculated)	%b	busy
Queue wait time		avque_t (ms)	avwait(ms)
Device service time	rtime		avserv(ms)
Number of reads and writes	reads	r/s	r+w/s
	writes	w/s	
Number of blocks read and written per second	read	blk/s (Kbytes)	blk/s (bytes)
	write	blk/s (Kbytes)	
Percentage of time system is idle waiting for I/O from this disk		%w	
Percentage of time system is idle waiting for I/O from any disk			%wio

Available I/O Statistics

The overhead image above shows some of the most important I/O subsystem data available from sar and iostat. The data, while coming from the same origin in the kernel, may be reported differently by the tools. Be sure you take this into account when comparing data from the different tools.

```
# sar -d
```

```
SunOS grouper 5.6 Generic_105181-03 sun4u      05/25/99
```

08:58:48	device	%busy	avque	r+w/s	blk/s	avwait	avserv
08:58:50	fd0	0	0.0	0	0	0.0	0.0
	nfs1	0	0.0	0	0	0.0	0.0
	nfs2	0	0.0	0	7	0.5	15.5
	nfs3	0	0.0	0	18	0.0	7.6
	sd0	6	0.1	4	97	0.0	32.1
	sd0,a	2	0.1	1	21	0.0	46.4
	sd0,b	4	0.1	3	76	0.0	26.7
	sd0,c	0	0.0	0	0	0.0	0.0
	sd0,d	0	0.0	0	0	0.0	0.0

```
#
```

Some fields in this output are:

- `device` – Name of the disk device being monitored.
- `%busy` – Percentage of time the device spent servicing a transfer request.
- `avwait` – Average time, in milliseconds, that transfer requests wait idly in the queue (measured only when the queue is occupied).
- `avserv` – Average time, in milliseconds, for a transfer request to be completed by the device (for disks, this includes seek, rotational latency, and data transfer times).
- `r+w/s` – Number of read and write transfers to the device per second.
- `blks/s` – Number of 512-byte blocks transferred to the device per second.

```
# sar -u
```

```
SunOS rafael 5.7 Generic sun4u      06/21/99

00:00:00    %usr    %sys    %wio    %idle
08:00:00      0        0        0       99
08:20:02      3        2       13       82
08:40:02      3        2       12       83
09:00:02      1        1        5       93
09:20:02      4        2        9       85

Average       1        0        1       97
#
```

The I/O statistic shown in this output is:

- `%wio` – The percentage of time the processor is idle and waiting for I/O completion

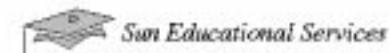
```
# iostat -x
```

device	extended device statistics								
	r/s	w/s	kr/s	kw/s	wait	actv	svc_t	%w	%b
fd0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0	0
sd0	0.6	0.3	4.7	5.0	0.0	0.0	49.3	0	1
sd6	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0	0
nfs1	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0	0
nfs2	0.0	0.1	0.4	2.9	0.0	0.0	25.3	0	0
nfs3	0.1	0.0	1.2	0.0	0.0	0.0	7.6	0	0
nfs545	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0	0

```
#
```

This output contains the following fields:

- %b – Percentage of time the disk is busy; that is, transactions in progress and commands are active on the drive.
- svc_t – The average I/O service time. Normally, this is the time taken to process the request once it has reached the front of the queue, however, in this case, it is actually the response time of the disk.
- r/s – The number of reads per second.
- w/s – The number of writes per second.
- kr/s – The number of Kbytes read per second.
- kw/s – The number of Kbytes written per second.
- %w – Percentage of time the queue is not empty; that is, the percentage of time transactions wait for service.



Available I/O Data

`ks_disks.nread` – Number of bytes a device has read
`ks_disks.nwritten` – Number of bytes a device has written
`ks_disks.reads` – Number of read operations
`ks_disks.rlentime` – Average length of time a transfer request takes to run (actual device service time)
`ks_disks.rtime` – Amount of time the device is active
`ks_disks.wlentime` – Average length of time a transfer request has to spend waiting to be serviced (pre-interface time)
`ks_disks.writes` – Number of write operations

Available I/O Data

The overhead image shows the definition of the kernel fields reported in the tuning reports.

iostat Service Time

The `iostat` service time is not what it appears to be. It is the sum of the times taken by a number of items, only some of which are due to disk activity. The `iostat` service time is measured from the moment that the device driver receives the request until the moment it returns the information. This includes driver queueing time as well as device activity time (true service time).

If the drive does not support tagged queuing, multiple requests are not reordered and each request is dealt with serially. Time spent by one request waiting for any previous requests to complete will affect its reported service time.

For example, suppose an application sent five requests almost simultaneously to a disk. Service times and response times may be:

Request Number	Service Time (Each Request)	Response Time
1	29.65 ms	29.65 ms
2	15.67 ms	45.32 ms
3	13.81 ms	59.13 ms
4	15.05 ms	74.18 ms
5	14.52 ms	88.70 ms

You can see a range of “reasonable” service times, as seen by the device driver. As the reported service times gradually increase, later requests must wait for the earlier ones to complete.

The average `iostat` service time is the sum of all response times divided by five: $(29.65 + 45.32 + 59.13 + 74.1 + 88.70) \div 5 = 49.50$ ms. The `iostat` report would include in its measure the time request 5 spent waiting for requests 1 to 4 to clear. In this particular scenario, it is possible to get from the reported figure back to an average service time for each request using the following formula:

Mean true service time is $2 \times (\text{reported service time}) \div (n+1)$, where n is the number of simultaneous requests. Thus the mean service time in the example is $2 \times 49.50 \div 6 = 16.50$.

Disk Monitoring Using SE Toolkit Programs

`siostat.se`

The `siostat.se` program is a version of `iostat` that shows the real definition of service times, which is the time it takes for the first command in line to be processed.

```
# /opt/RICHpse/bin/se /opt/RICHpse/examples/siostat.se
16:37:21      -----throughput-----  -----wait queue-----  -----active queue-----
disk      r/s    w/s    Kr/s    Kw/s  qlen res_t svc_t %ut qlen res_t svc_t %ut
c0t6d0    0.0    0.0    0.0     0.0  0.00  0.00  0.00    0 0.00  0.00  0.00  0.00   0
c0t0d0    7.0    0.0   49.0    0.0  0.00  0.00  0.00    0 0.05  7.49  7.49   5
c0t0d0s0  7.0    0.0   49.0    0.0  0.00  0.00  0.00    0 0.05  7.48  7.48   5
c0t0d0s1  0.0    0.0    0.0     0.0  0.00  0.00  0.00    0 0.00  0.00  0.00  0.00   0
c0t0d0s2  0.0    0.0    0.0     0.0  0.00  0.00  0.00    0 0.00  0.00  0.00  0.00   0
c0t0d0s3  0.0    0.0    0.0     0.0  0.00  0.00  0.00    0 0.00  0.00  0.00  0.00   0
fd0       0.0    0.0    0.0     0.0  0.00  0.00  0.00    0 0.00  0.00  0.00  0.00   0
^C#
```

`xio.se`

The `xio.se` program is a modified `xiostat.se` that tries to determine whether accesses are mostly random or sequential by looking at the transfer size and typical disk access time values. Based on the expected service time for a single random access read of the average size, results above 100 percent indicate random access, and those under 100 percent indicate sequential access.

```
# /opt/RICHpse/bin/se /opt/RICHpse/examples/xio.se
                                                 extended disk statistics
disk      r/s    w/s    Kr/s    Kw/s  wtq actq  wtres actres  %w  %b  svc_t %rndsvc
c0t6d0    0.0    0.0    0.0     0.0  0.0  0.0    0.0    0.0    0    0    0.0    0.0
c0t0d0    9.0    7.0   72.0    56.0 0.0  0.4    0.0   25.5    0   16   10.1   65.4
c0t0d0s0  4.0    0.0   32.0    0.0  0.0  0.0    0.0    7.5    0    3    7.5   48.7
c0t0d0s1  5.0    7.0   40.0    56.0 0.0  0.4    0.0   31.4    0   13   10.9   70.7
c0t0d0s2  0.0    0.0    0.0     0.0  0.0  0.0    0.0    0.0    0    0    0.0    0.0
c0t0d0s3  0.0    0.0    0.0     0.0  0.0  0.0    0.0    0.0    0    0    0.0    0.0
fd0       0.0    0.0    0.0     0.0  0.0  0.0    0.0    0.0    0    0    0.0    0.0
^C#
```

xiostat.se

The xiostat.se program is a basic iostat -x clone that prints the name of each disk in a usable form.

```
# /opt/RICHpse/bin/se /opt/RICHpse/examples/xiostat.se
                                extended disk statistics
disk      r/s    w/s    Kr/s    Kw/s   wait   actv   svc_t   %w   %b
c0t6d0    0.0    0.0    0.0     0.0    0.0    0.0    0.0    0     0
c0t0d0    4.0    3.0   32.0    32.0   0.0    0.1   13.9    0     6
c0t0d0s0  4.0    0.0   32.0    0.0    0.0    0.0    7.8    0     3
c0t0d0s1  0.0    3.0    0.0    32.0   0.0    0.1   21.9    0     3
c0t0d0s2  0.0    0.0    0.0    0.0    0.0    0.0    0.0    0     0
c0t0d0s3  0.0    0.0    0.0    0.0    0.0    0.0    0.0    0     0
fd0       0.0    0.0    0.0     0.0    0.0    0.0    0.0    0     0
^C#
```

iomonitor.se

The iomonitor.se program is like iostat -x, but it does not print anything unless it sees a slow disk. An additional column headed delay prints out the total number of I/Os multiplied by the service time for the interval. This is an aggregate measure of the amount of time spent waiting for I/Os to the disk on a per-second basis. High values could be caused by a few very slow I/Os or many fast ones.

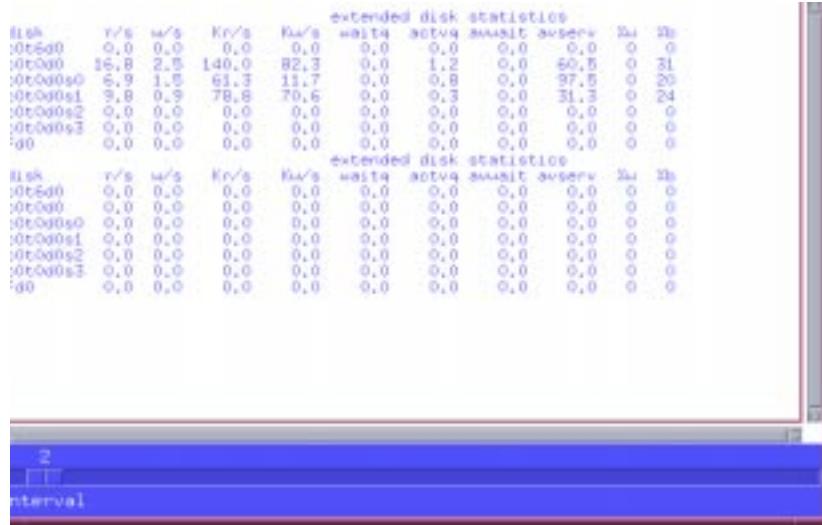
iost.se

The iost.se program is iostat -x output that does not show inactive disks. The default is to skip 0.0 percent busy.

```
# /opt/RICHpse/bin/se /opt/RICHpse/examples/iost.se
                                extended disk statistics
disk      r/s    w/s    Kr/s    Kw/s   wait   actv   svc_t   %w   %b
c0t0d0    0.0    1.0    0.0     8.0    0.0    0.0   20.1    0     2
c0t0d0s0  4.0    1.0   32.0    8.0    0.0    0.0    8.2    0     4
TOTAL w/s      2.0          16.0
r/s        4.0          32.0
^C#
```

xit.se

The **xit.se** program is the eXtended Iostat Tool. It produces the same as running **iostat -x** in a **cmdtool**. It uses a sliding bar to interactively set the update rate and to start/stop/clear the output.



disks.se

The **disks.se** program prints out the disk instance to controller/target mappings and partition usage.

```
# /opt/RICHPSe/bin/se /opt/RICHPSe/examples/disks.se
se thinks MAX_DISK is 8
kernel  -> path_to_inst  -> /dev/dsk part_count [fstype mount]
sd6      -> sd6          -> c0t6d0          0
sd0      -> sd0          -> c0t0d0          3
                  -> c0t0d0s0        ufs   /
                  -> c0t0d0s1        swap  swap
                  -> c0t0d0s3        ufs   /cache
sd0,a    -> sd0,a        -> c0t0d0s0        0
sd0,b    -> sd0,b        -> c0t0d0s1        0
sd0,c    -> sd0,c        -> c0t0d0s2        0
sd0,d    -> sd0,d        -> c0t0d0s3        0
fd0      -> fd0          -> fd0            0
#
```

Check Your Progress

Before continuing on to the next module, check that you are able to accomplish or answer the following:

- Demonstrate how I/O buses operate
- Describe the configuration and operation of a SCSI bus
- Explain the configuration and operation of a Fibre Channel bus
- Demonstrate how disk and tape operations proceed
- Explain how a storage array works
- List the RAID levels and their uses
- Describe the available I/O tuning data

Think Beyond

Some questions to ponder are:

- Why are there only eight luns on a SCSI target?
- What would happen if the SCSI initiator was not the highest priority device on the bus? How could you tell?
- What devices besides a disk drive might tagged queueing benefit?
- What special considerations are there in looking at tuning reports from a storage array?

Objectives

Upon completion of this lab, you should be able to:

- Identify the properties of SCSI devices on a system
- Calculate the time an I/O operation should take
- Compare calculated I/O times with actual times

Tasks

SCSI Device Characteristics

Use the following steps to determine I/O subsystem characteristics:

1. Locate your boot drive and CD-ROM drive. Use `prtconf -v` to identify the following characteristics for each:

Target _____

Device type _____

Speed _____

Tagged queueing? _____

Wide SCSI? _____

Target _____

Device type _____

Speed _____

Tagged queueing? _____

Wide SCSI? _____

2. Using `prtconf -v`, determine the characteristics of the first SCSI bus in your system. Use Appendix A and estimate the adapter speed from the speeds of the devices on the bus.

SCSI host adapter

Type _____

Speed _____

3. Using `prtvtoc`, determine the number of cylinders that the drive has.

Physical _____

Logical _____

4. Using the diagram and tables in Module 8, calculate the time an I/O operation should take on the device. Assume that the interrupt process does not add any time.

If no other data are available, assume an 8 Kbyte transfer, a seek time of 10 ms, and an ITR of 5 Mbytes/sec.

Arbitration _____

Data transfer _____

Seek time _____

Rotation _____

Internal transfer _____

Status _____

Total _____

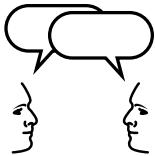
5. Run an iostat report for the device. Do the figures (for a lightly loaded system) seem similar?

Objectives

Upon completion of this module, you should be able to:

- Describe the vnode interface layer to the file system
- Explain the layout of a UFS partition
- Describe the contents of an `inode` and a directory entry
- Describe the function of the superblock and cylinder group blocks
- Understand how allocation is performed in the UFS
- List the UFS caches and how to tune them
- Describe how to use the other UFS tuning parameters

Relevance



Discussion – The following questions are relevant to understanding the content of this module:

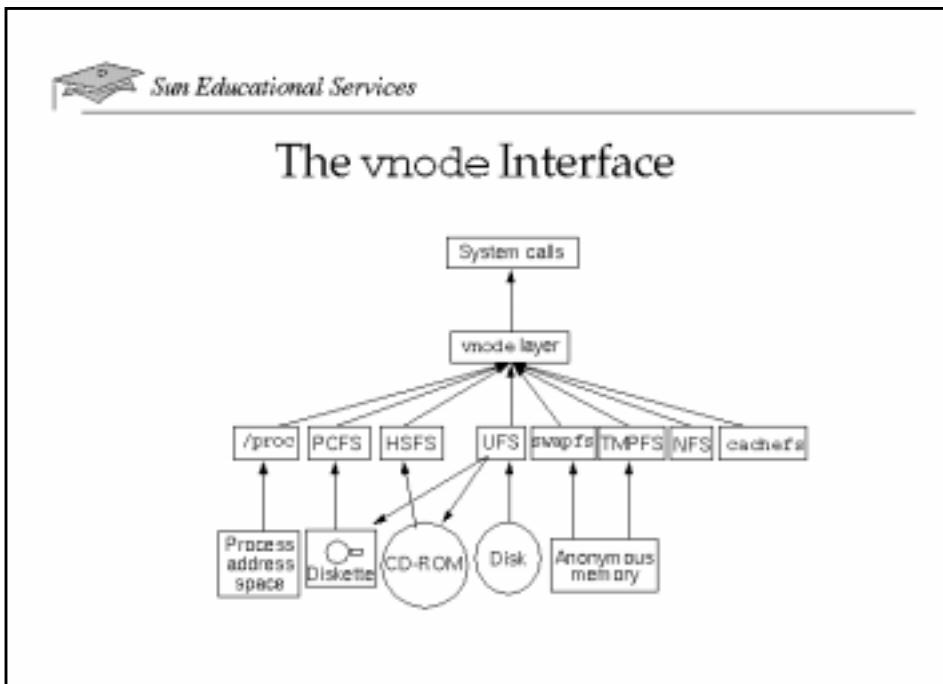
- Why would you want to tune a UFS partition?
- What might be tunable in a partition?
- Is it more important to tune the partition or the drive?
- Does the use of storage arrays change the way a partition should be tuned?

Additional Resources



Additional resources – The following references can provide additional details on the topics discussed in this module:

- M. McKusick, W. Joy, S. Leffler, and R. Fabry, “A Fast File System for UNIX,” *ACM Transactions on Computer Systems*, 2(3) August 1984.
- Cockcroft, Adrian, and Richard Pettit. 1998. *Sun Performance and Tuning*. Palo Alto: Sun Microsystems Press/Prentice Hall.
- Wong, Brian L. 1997. *Configuration and Capacity Planning for Solaris Servers*. Palo Alto: Sun Microsystems Press/Prentice Hall.
- Graham, John. 1995. *Solaris 2.x Internals*. McGraw-Hill.
- Mauro/McDougall. 1999. *Solaris Architecture and Internals*. Sun Microsystems Press/Prentice Hall.
- Man pages for the various commands (newfs, tunefs, fstyp, mkfs) and performance tools (sar, vmstat, iostat, netstat).



The vnode Interface

One of the major design paradigms of the UNIX operating system is that everything looks like a file. In SunOS, this is true even in most of the kernel.

The SunOS operating system has the ability to access SunOS file systems on local disks, remote file systems via NFS, and many other types of file systems, such as the High Sierra File System and MS-DOS file systems. This is accomplished using vnodes.

The vnode provides a generic file interface at the kernel level, so the calling component does not know what sort of file system it is dealing with. Since it is this transparent in the kernel, it is also transparent to the user.

The vnode interface:

- Provides an architecture that multiple file system implementations can use within SunOS
- Separates the file system dependent functionality from the file system independent functionality
- Specifies a well-defined interface for the file system dependent and file system independent interface layers

The different file systems are manipulated through the `vfs` (virtual file system) structure, which composes the mounted file system list. A `vfs` is like a `vnode` for a partition.

All files, directories, named pipes, device files, doors, and so on are manipulated using the `vnode` structure.



The Local File System

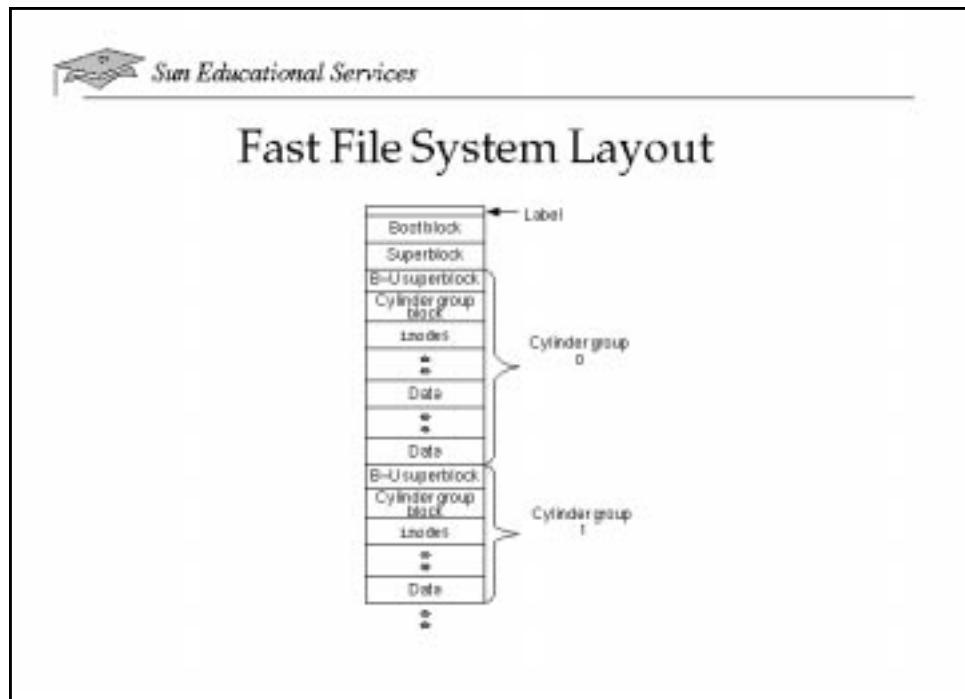
- A disk drive must be divided into one or more partitions or slices.
- Each partition can contain only one file system.
- Files are accessed in fixed-length blocks on the disk.
 - Four-Kbyte and 8-Kbyte blocks are available
 - Four-Kbyte blocks are not supported for Ultra systems
- Information about a file is kept in an `inode`.
- A directory points to an `inode` for every named object in a partition.

The Local File System

The SunOS UFS file system is implemented as a derivative of the Berkeley BSD Fast Fat File system. It has the following characteristics:

- A disk drive is divided into one or more slices or partitions. Each slice can contain one and only one file system.
- Files are accessed in fixed-length blocks on the disk, almost invariably 8 Kbytes long. While 4-Kbyte blocks are an option, they are not supported on UltraSPARC systems.
- Information about a file is kept in an `inode`.
- A directory references an `inode` for each named object on disk.

From the user level, any file is viewed as a stream of bytes. The underlying hardware manages the file data as blocks of bytes. The minimum size the hardware can write or read from disk is a sector of 512 bytes.



Fast File System Layout

This diagram shows the order `mksfs` (called by `newfs`) lays out the file system on disk.

Disk Label

The disk label is the first sector of a partition. It is only present for the first partition on a drive. It contains the drive ID string, a description of the drive geometry, and the partition table, which describes the start and extent of the drive's partitions or slices.

Boot Block

The remainder of the first 8-Kbyte block in the partition contains the boot block if this is a bootable partition. Otherwise the space is unused.

Superblock

The superblock is the second 8-Kbyte block in the partition, starting at sector 16. It contains two types of data about the partition: static, descriptive data and dynamic, allocation status data. It is used to describe to UFS the current configuration and state of the partition.

The contents of the superblock can be viewed with the `fstyp -v` command.

Cylinder Groups

The remainder of the partition is divided into cylinder groups (cg). A cylinder group has a default size of 16 cylinders (in Solaris 2.6 and later environments, it is calculated based on partition size), regardless of the disk geometry. It can be made significantly larger.

The cylinder group has four components: backup superblock, cylinder group block, inode table, and data blocks.

Backup Superblock

Each cylinder group contains a backup superblock, usually as the first 8-Kbyte block in the cg. The backup superblock is written when the partition is created and is never written again. It contains the same static descriptive data that the primary superblock does, which is used to rebuild the primary superblock if it becomes damaged. The first backup superblock starts at sector 32.

Cylinder Group Block

The cylinder group block (cgb) is the 8-Kbyte block immediately following the backup superblock. It contains bit maps and summaries of the available and allocated resources, such as `inodes` and data blocks, in the cylinder group.

Each cylinder group's summary information (16 bytes) is also kept in the superblock for use during allocation.

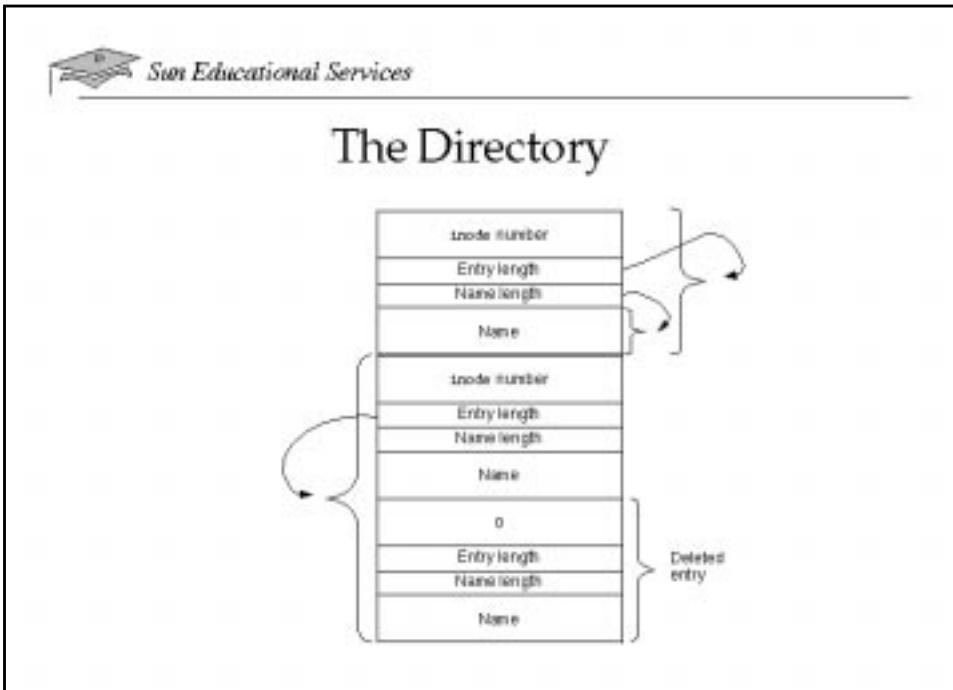
The inode Table

The next part of the cylinder group contains a share of the partition's inodes, just after the cgb. The total number of inodes in the partition is calculated based on the newfs -i operand and the size of the partition. The resulting total number of inodes is then divided by the number of cylinder groups, and that number of 128-byte inodes is then created in each cylinder group.

A bit map in the cgb tracks which inodes are allocated and free.

Data Blocks

The remainder of the partition is divided into data blocks. Their availability is also tracked in bit maps in the cgb.



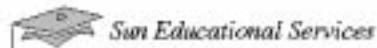
The Directory

The directory is the fundamental data structure used to locate files in a partition. Each directory is pointed to by an `inode`, which in turn points to other `inodes`.

Directories are allocated in 512-byte (one sector) blocks. Each directory entry consists of four fields. The first is the inode number of the file the entry represents, the second the total length of the entry, the third the length of the file name, and the fourth the variable length file name itself. The directory structure is described in `/usr/include/sys/dirent.h`.

Any holes in the directory caused by deletion, and the leftover space at the end of a directory block, are accounted for by having the previous entry's record length include the unused bytes. Deleted entries at the beginning of a 512-byte block are recognized by an `inode` number of 0.

New entries are allocated on a first-fit basis, using the holes in the directory structure.



Directory Name Lookup Cache (DNLC)

- Searching for a name in a directory is expensive.
- DNLC is a cache of the most recently referenced directory entry names and their associated vnodes.
- It is used by UFS and NFS.
- *ncsize* determines the size of DNLC.
- The maximum name size is 31 characters.
- *vmstat -s* and *sar* display DNLC statistics.

Directory Name Lookup Cache (DNLC)

Searching for a name in a directory is expensive: many small blocks might need to be read, and each must be searched, making this a very expensive operation.

The DNLC is used by both UFS and NFS.

To save time, each used directory entry is cached in the directory name lookup cache. Prior to the Solaris 7 OS, the directory component name (between the '/')s) must be 31 bytes long or smaller to be cached (the actual field is 32 characters long but the last character is a termination character); if it is not, the system must go to disk every time the long name is requested. The Solaris 7 OS has no length restriction.

The *ncsize* parameter determines the size of DNLC. It is scaled from the *maxusers* parameter and should be the same size as the UFS inode cache discussed later in this module.

The DNLC caches the most recently referenced directory entry names and the associated vnode for the directory entry. Names are searched for in the cache by using a hashing algorithm. The hashing algorithm uses three values: the entry name, the length of the name, and a pointer to the vnode of the parent directory. If the requested entry is in the cache, the vnode will be returned.

The number of name lookups per second is reported as *namei/s* by the sar -a command.

```
# sar -a
```

```
SunOS rafael 5.7 Generic sun4u      06/22/99

00:00:01  iget/s namei/s dirbk/s
01:00:01      0      3      0
02:00:02      0      3      0
...
...
15:20:02      0      6      2
Average       1     11      1
#
```

One field in this output is:

- *namei/s* – The number of file system path searches per second. If *namei* does not find a directory name in the DNLC, it calls *iget* to get the inode for either a file or directory. Most *iget* calls are the result of DNLC misses.



The inode Cache

- Contains an entry for every open inode
 - Holds as many closed inodes as possible
- Locates entries through the DNLC
- Grows in size to hold all open files
 - Can exceed your setting
 - If it does exceed your setting, closed (unused) inodes are flushed
- When closed inodes are cached, reuses their data pages
 - If flushed (removed from cache), loses the pages

The inode Cache

The inode cache and the DNLC are closely related. For every entry in the DNLC there will be an entry in the inode cache. This allows the quick location of the inode entry, without needing to read the disk.

As with most any cache, the DNLC and inode caches are managed on a least recently used (LRU) basis; new entries replace the old. But, just as in main storage, entries that are not being currently used are left in the cache on the chance that they might be used again.

If the cache is too small, it will grow to hold all of the open entries. Otherwise, closed (unused) entries remain in the cache until they are replaced. This is beneficial, for it allows any pages from the file to remain available in memory. This allows files to be opened and read many times, with no disk I/O occurring.

The *ufs_ninode* parameter determines the size of the inode cache. It is scaled from *maxusers* and should be the same size as the DNLC.



The inode Cache

- inode cache size is set by *ufs_ninode*.
- You can use netstat -k to display inode cache statistics.
- *maxsize* is the variable equal to *ufs_ninode*.
- A *maxsize_reached* value higher than *maxsize* indicates the number of active inodes has exceeded cache size.
- In this case, you should increase *ufs_ninode*.
- You can add line to /etc/system.

```
set ufs_ninode=5000
```

You can see inode cache statistics by using netstat -k to display the raw kernel statistics information.

```
# netstat -k
kstat_types:
raw 0 name=value 1 interrupt 2 i/o 3 event_timer 4

segmap:
fault 2763666 faulta 0 getmap 4539376 get_use 973 get_reclaim 4023294 get_reuse 508075
...
...
inode_cache:
size 1922 maxsize 4236 hits 41507 misses 202147 kmem allocs 32258 kmem frees 30336
maxsize reached 5580 puts at frontlist 189314 puts at backlist 17404
queues to free 0 scans 139285773 thread idles 199953 lookup idles 0 vget idles 0
cache allocs 202147 cache frees 222194 pushes at close 0
...
...
buf_avail_cpu0 0

#
```



DNLC and inode Cache Management

- DNLC and inode caches should be the same size.
 - Each DNLC entry has an inode entry.
 - Default sizes are $4 \times (\maxusers + \max_nprocs) + 320$.
 - inode cache size is set by `ufs_ninode`.
 - DNLC cache size is set by `ncsize`.
 - Set them high (5,000 or more) on NFS™ servers
 - Try to keep the DNLC hit rate above 90 percent

DNLC and inode Cache Management

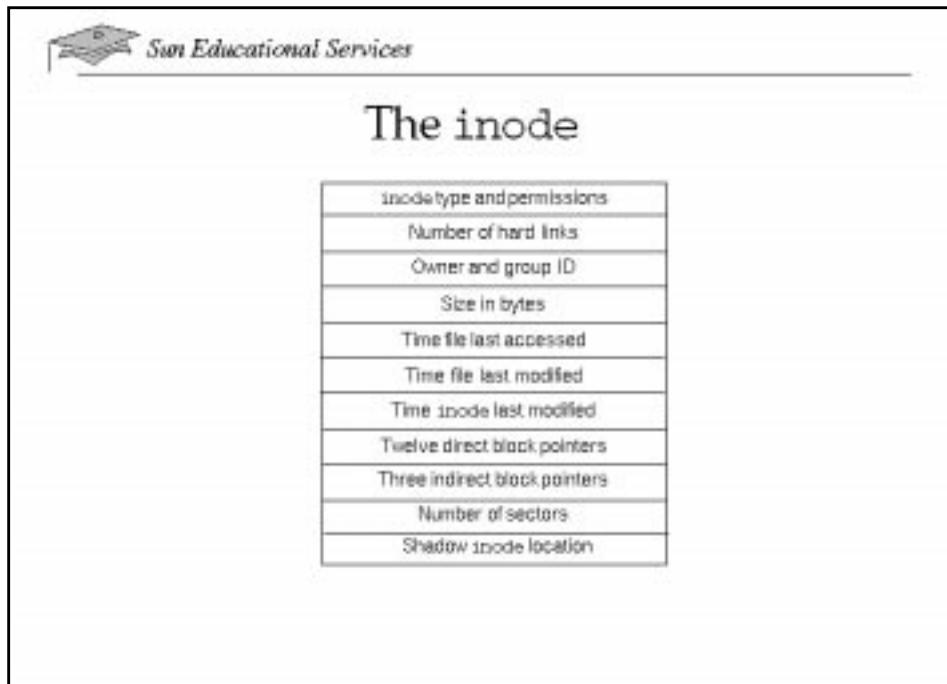
As mentioned earlier, the DNLC and inode caches should be the same size since each DNLC entry has an inode cache entry.

The default sizes for both caches are $4 \times (\maxusers + \max_nprocs) + 320$ entries. Cache sizes as large as 8,000 entries or more are not unreasonable for NFS servers.

Entries removed from the inode cache are also removed from the DNLC. Since the inode cache caches only UFS data (inodes) and the DNLC caches both NFS (rnodes) and UFS (inodes) data, the ratio of NFS to UFS activity should be considered when modifying `ncsize`.

`vmstat -s` will display the DNLC hit rate and lookups that failed because the name was too long. The hit rate should be 90 percent or better. The sar options used with the DNLC and inode caches are covered near the end of this module.

```
# vmstat -s
    0 swap ins
    0 swap outs
    0 pages swapped in
    0 pages swapped out
3795975 total address trans. faults taken
1019061 page ins
    329506 page outs
1273507 pages paged in
1148106 pages paged out
135027 total reclaims
    89814 reclaims from free list
        0 micro (hat) faults
3795975 minor (as) faults
1014039 major faults
    470479 copy-on-write faults
1310922 zero fill page faults
6602052 pages examined by the clock daemon
        905 revolutions of the clock hand
2189171 pages freed by the clock daemon
    19286 forks
        3981 vforks
        24873 execs
297088833 cpu context switches
398206699 device interrupts
    6882203 traps
879394072 system calls
15665385 total name lookups (cache hits 95%)
    1129529 user    cpu
        539059 system  cpu
133157029 idle    cpu
    1307463 wait    cpu
#
```



The inode

The diagram in the overhead image above shows most of the contents of the on-disk `inode` structure.

The file size is an 8-byte integer (long) field. As of the SunOS 5.6 operating system, the largest file size supported by UFS is 1 Tbyte. To use more than 2 Gbytes, programming changes are needed in most applications. (See the man page for `largefiles(5)` for more information.)

The number of sectors is kept as well as the file size due to UFS support for sparse or "holey" files, where disk space has not been allocated for the entire file. It can be seen with the `ls -s` command, which also gives the actual amount of disk space used by the file.

The shadow inode pointer is used with access control lists (ACLs).

inode *Time Stamp Updates*

There are three time stamp fields in the `inode`. For certain workloads, constant updates of these time stamps are not necessary, and can cause performance problems. Updates of two of these three time stamps can be turned off.

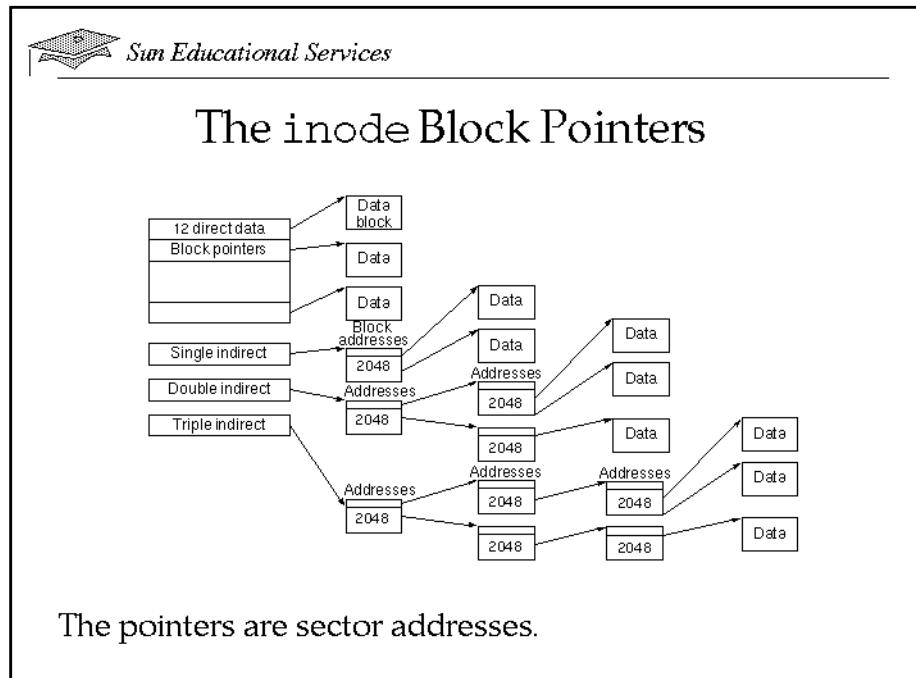
- File access time (`atime`) – Every time a file is accessed, whether for read or write, the access time stamp is updated. Normally, this time stamp is used to ensure that files being used are not migrated to tape. For some applications, such as web and network news servers, this last access date is not useful.

To turn off these updates, in the Solaris 7 environment, mount the partition with the `noatime` option. No access time updates will be made for any file in the partition. See the man page for `mount_ufs(1M)` for more detail.

- File modification time (`mtime`) – Every time the file is modified, the file modification time stamp is updated. This allows incremental backups to be performed: it is easy to tell which files need to be backed up. Some files, such as DBMS tablespaces, are constantly written to but are backed up using other criteria. Constant updates of these files result in unnecessary I/O.

To turn off these updates, set the sticky bit for the file using, for example, `chmod u+t`. The file's execute permission bit must be off. See the man page for `sticky(4)` for more information on the sticky bit.

- inode modification time (`ctime`) – The `inode` modification time is set when the `inode` is changed, such as by `chmod` or `chown`. These updates cannot be turned off.



The inode Block Pointers

The inode contains two sets of block pointers, direct and indirect.

The direct pointers point to one data block, usually 8 Kbytes. With 12-block pointers, this covers 96 Kbytes of the file.

If the file is larger than 96 Kbytes, the indirect block pointers are used. The first indirect block points to a data block, which has been divided into 2048 pointers to data blocks. The use of this indirect block covers an additional 16 Mbytes of data.

If the file is larger than 16 Mbytes, the second indirect pointer is used. It points to a block of 2048 addresses, each of which points to another indirect block. This allows for an additional file size of 32 Gbytes.

The third pointer points to a third level indirect block, which allows an additional 64 Tbytes to be addressed. However, remember that the maximum file size is limited to 1 Tbyte at this time.



UFS Buffer Cache

- Holds *inodes*, cylinder group blocks, and indirect blocks only
- Has *bufhwm* as the only parameter specified in Kbytes
 - A default of 0 allows up to 2 percent of main memory to be used.
 - This may be too much on very large systems.
 - You can usually reduce it to a few Mbytes.
- Uses *sysdef* to determine the current setting

UFS Buffer Cache

The UFS buffer cache (also known as metadata cache) holds *inodes*, cylinder groups, and indirect blocks only (no file data blocks).

The default buffer cache size allows up to 2 percent of main memory to be used for caching the file system metadata just listed. The size can be adjusted with the *bufhwm* parameter, which is specified in Kbytes.

The default may be too much on systems with very large amounts of main memory. You can reduce it to a few Mbytes with an entry in the */etc/system* file.

You can use the *sysdef* command to check the size of the cache. Read and write information is reported by *sar -b*, as *lreads* and *lwrites*.

The UFS buffer cache is sometimes confused with the *segmap* cache, which is used for managing application data file requests.

The following partial output of the *sysdef* command displays the size of the buffer cache (*bufhwm*):

```
# sysdef
*
* Hostid
...
...
* Tunable Parameters
*
1196032      maximum memory allowed in buffer cache (bufhwm)
...
...
#
```

The following displays partial output of the **sar -b** command:

```
# sar -b

SunOS rafael 5.7 Generic sun4u      06/22/99

00:00:01 bread/s lread/s %rcache bwrit/s lwrit/s %wcache pread/s pwrit/s
01:00:01      0      1     100      0      1     74      0      0
02:00:02      0      1     100      0      1     74      0      0
...
...
15:00:01      0      1      93      0      1      56      0      0
Average       0      1      95      0      1      67      0      0
#
```

In this output are:

- lread/s – Average number of logical reads per second from the buffer cache.
- lwrit/s – Average number of logical writes to the buffer cache per second.

 Sun Educational Services

Hard Links



- Work only in the same partition
- Have no limit to the number
- Have no back pointers
 - Use `find -inum` to locate the other names

Hard Links

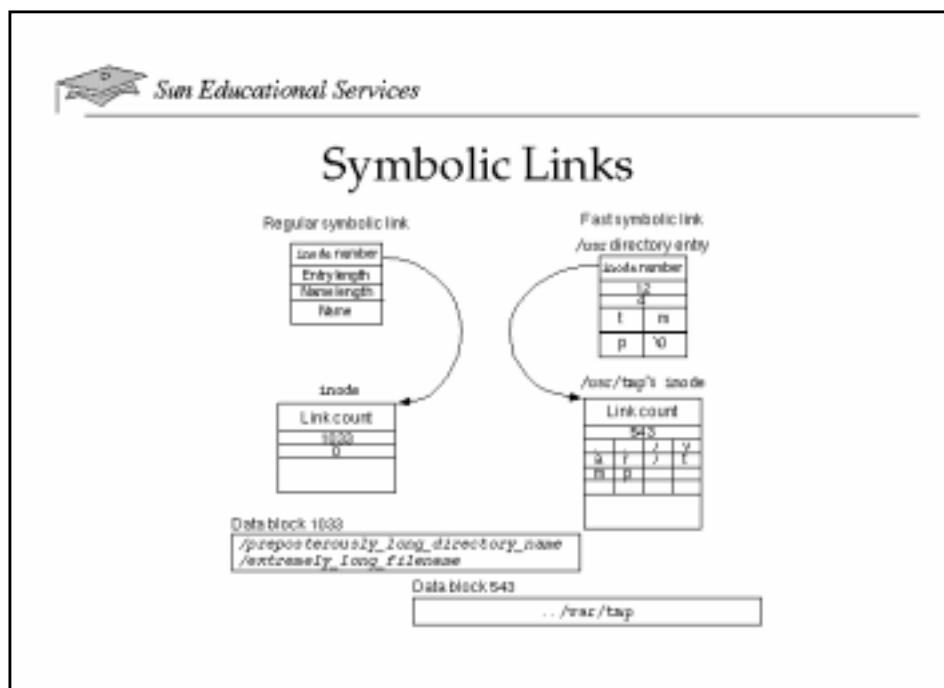
The `inode` number contained in the directory entry creates a *hard link* between the file name and the `inode`.

A file always has a single `inode`, but multiple directory entries can point to the same `inode`. The reference count in the `inode` structure is the number of hard links referencing the `inode`. When the last link is deleted (unlinked), the `inode` can be deallocated and the associated file space freed.

There are no back pointers from the `inode` to any directory entries.

An entry in the DNLC is made for each hard link, but the same `inode` cache entry is used for each.

Since the hard links use `inode` pointers, they can be used only within a partition. To reference a file in another partition, symbolic or soft links must be used.



Symbolic Links

SunOS also supports a link to a file that is implemented by pointing to a file that contains a path name, either absolute or relative. This is a soft or symbolic link and can point to a file or directory on a different file system and file system type.

If the number of symbolic links referenced during a translation is greater than 20, it is assumed that a loop has been created and an error is returned.

A *fast symbolic link* is used when the path name that is being linked is less than or equal to 56 characters. The 56 characters represent the fourteen 4-byte unused direct and indirect block pointers, where the file name referenced by the link is placed.



Allocating Directories and inodes

<code>d d i i b b</code>	<code>d d i i b b b</code>	<code>d d i i b b b</code>	?
--------------------------	----------------------------	----------------------------	---

UFS tries to provide even performance regardless of partition utilization.

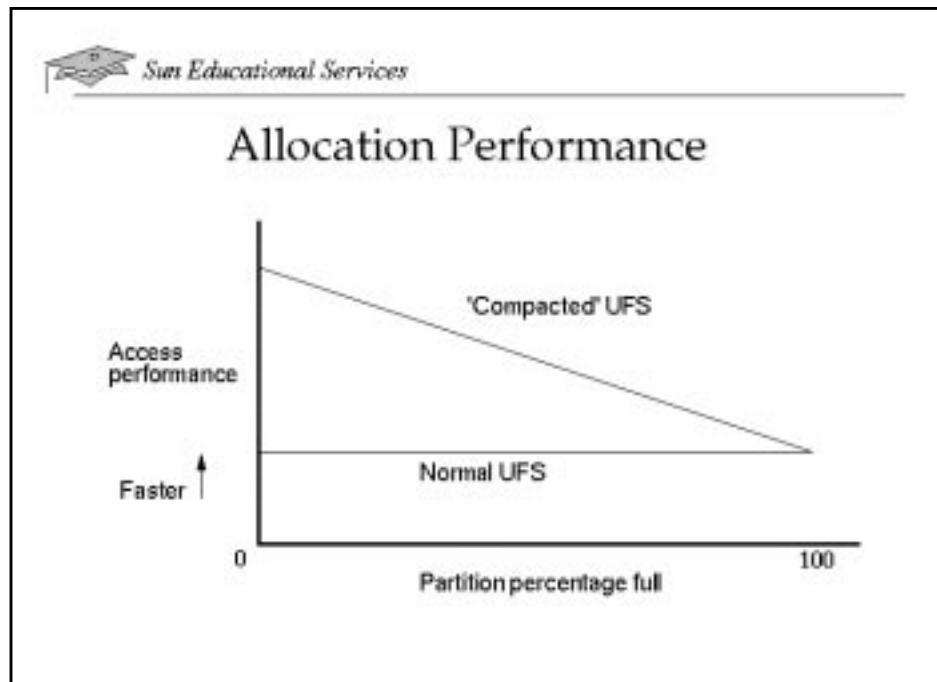
- Directories are allocated in a cylinder group (cg) containing a larger than average number of free inodes.
 - Ties are broken by the fewest directories.
- File inodes are allocated to the same cg as their directory.
- Data blocks are allocated in the same cg as their inode.

Allocating Directories and inodes

UFS tries to evenly allocate directories, inodes, and data blocks across the cylinder groups. This provides a constant performance (discussed on the next page) and provides a better chance for related information to be kept close by.

The determination of which cylinder group to use is made from the cylinder group allocation summaries kept in the primary superblock.

Allocated data is clustered near the front of the cylinder group.



Allocation Performance

Many non-UFS file systems allocate from the front of the partition to the rear, regularly compressing the partition to move data to the front.

This "compacted" approach was rejected during the design of UFS in favor of allocation across cylinder groups.

The benefit of the UFS approach is that the disk arm would move, on average, half of the partition regardless of how full the partition is. In a compacted file system, the arm will move further and further, taking longer and longer, as the partition fills up. While still fast, it creates the perception that the system is slowing as the file system fills.



Allocating Data Blocks

Data block allocation is a five-step process.

1. Request the "ideal" block.
2. Take a block in the cylinder group.
3. Do quadratic rehash.
4. Look at all remaining cylinder groups.
5. Panic the system.

Allocating Data Blocks

Allocation of data blocks in a partition is done in five steps, as shown in the overhead image above.

The quadratic rehash algorithm is discussed on the next page.

The reason the last step invokes a panic is that the file system allocation data must be incorrect. If the partition was known to be full, the allocation request would have been failed immediately. Since it was not, the file system indicated that there was available space. When none was found, the file system was inconsistent. The panic will cause the system to run `fsck` during the reboot, correcting the information.



Quadratic Rehash

- Tries to avoid clumps of full cylinder groups
- Searches twice as far each time another cg is found full
- Stop quickly, which is important for performance
- Reserves free space to try to stop quickly
 - This is the "10 percent" reserved seen in df.
- In Solaris 2.6 OS, depends on partition size
 - The formula is $(64 \text{ Mbytes} \div \text{file system size}) \times 100$.
 - It runs from 10 percent ($\leq 640 \text{ Mbytes}$) to 1 percent ($\geq 6.4 \text{ Gbytes}$).

Quadratic Rehash

The quadratic rehash algorithm was developed to try to quickly locate random blocks of free space in a nearly full partition. It makes the assumption that if a cylinder group is full, the cylinder groups around it will be full too, so farther locations are more likely to have free space.

When the "ideal" cylinder group is full, the quadratic algorithm is used to try to locate a cylinder group with free space. The amount of free space is not considered.

The faster the quadratic rehash algorithm finds free space, the sooner it stops, so the partition has a certain amount of space set aside (as seen with the `df` command) that can only be allocated by the root user. In a large, nearly full partition, the algorithm can be very time consuming.

Until SunOS 5.6, this amount was 10 percent of the available space in the partition. In Solaris 2.6 operating systems and higher, the amount reserved depends on the size of the partition, running from 10 percent in small partitions to 1 percent in large ones.

The Algorithm

Assuming a partition of 33 cylinder groups, and a current cylinder group of 13, the algorithm would check other cylinder groups as follows:

Cylinder Group	Next Increment
13	1
14	2
16	4
20	8
28	16
11	32
10	64
...	

The algorithm would continue until it encountered a cylinder group that it had already checked. Then, the unchecked cylinder groups would be checked serially (the "brute force" step).



Fragments

- A full 8-Kbyte block is a waste for small files.
- Fragments use 1/8 block increments to extend a file.
- Fragments can be allocated by best-fit or full block.
- Fragment rules are:
 - Only the last block of the file can be fragmented.
 - Files larger than 96 Kbytes cannot use fragments.
 - Fragments must be contiguous within their block.
 - Fragments can be moved to remain contiguous.

Fragments

For very small files, the allocation of an entire 8-Kbyte block is a significant waste of space. UFS supports the suballocation of data blocks, called fragments.

The number of fragments per block can be set from 1 to 8 (the default) by using newfs.

Fragments can be allocated to a whole new 8-Kbyte block every time (optimization for time), or to the best-fit space available in an existing fragmented block (optimization for space). The optimization type can be changed by the tunefs utility.

The rules for fragment allocation are given in the overhead image above.



Logging File Systems

- Allocation requires lots of I/O
 - Cylinder group block, inode, and superblock updates are needed for one data block allocation
- Logging file systems write to the log; other writes are done later.
 - This takes advantage of write cancellation.
 - It has faster recovery because all changes are known and logged.
 - The log is usually a disk hot spot.
 - Some examples are VxFS and SunOS 5.7 UFS.

Logging File Systems

The ability of the UFS to allocate space as required and support files of practically unlimited size is the key to its flexibility.

The drawback of this flexibility is the amount of overhead required when allocating space. Every data block or fragment allocated requires at least three writes, which all should be done before the block is safe to use. In a busy system, this generates a large amount of I/O.

To avoid this overhead, a logging file system can be used. The logging file system writes a record of the allocation transaction to the file system log. Updates to the disk structures can be done at a later time.

This also provides a significant benefit in recovery time: the entire file system does not need to be checked. Only the log transactions need to be checked and rewritten if necessary.



Application I/O

- Application I/O usually goes through the `segmap` cache.
 - The `segmap` cache is a kernel memory segment.
 - The `segmap` cache is from 4 to over 90 Mbytes in size.
 - It is a fully associative, write-back cache.
The `segmap` cache is at the same kernel virtual address for every process.
 - The `segmap` cache contents differ depending on the process's file usage pattern.
 - Blocks in the cache might not be in memory.
 - Data access requires a system call and a copy of the data.

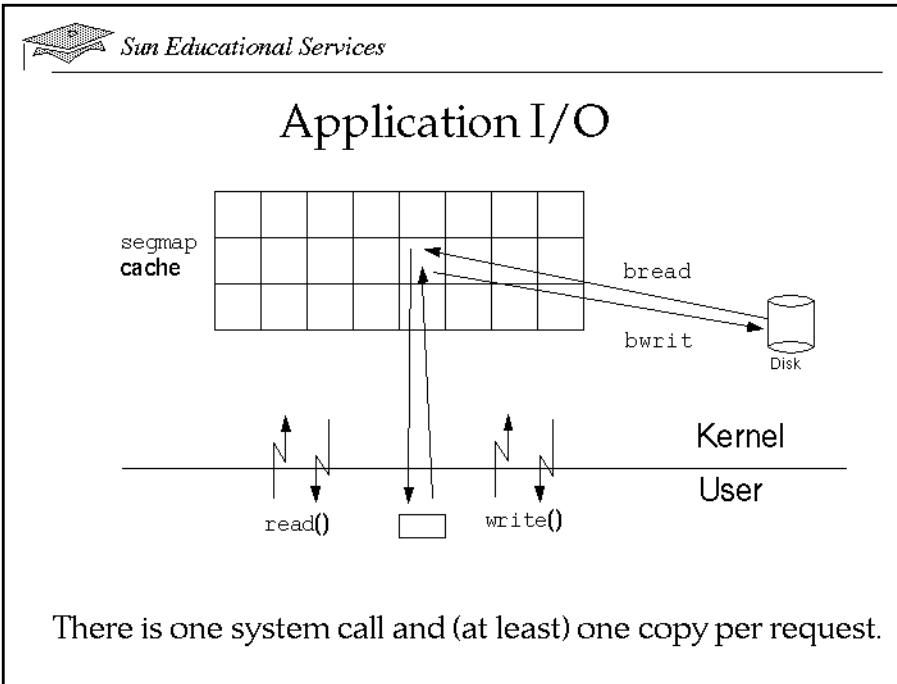
Application I/O

UFS application I/O is not usually done directly to a process address space. Instead, data is copied into a cache (the `segmap` cache) in the kernel, then copied into the user address space.

The `segmap` cache is a fully associative, write-back cache of varying sizes. It is at the same kernel virtual address for every process, but the contents differ depending on the process' file usage pattern.

The file system pages in the `segmap` cache may be shared with other processes' `segmap` caches, as well as other access mechanisms such as `mmap`.

Since the `segmap` cache is pageable, a block shown in the cache may not be present in memory. It will be page faulted back in if it is required again.



Access to the segmap cache is done by the `read` and `write` system calls.

The system call checks the cache to see if the data is present in the cache. If it is not, the segmap driver requests that the page be located and allocated to the selected cache's virtual address. If it is not already in memory, it is page faulted into the system. (For a full block write, the read is not necessary.) The specified data is then copied between the user region and the cache.

This can generate a significant amount of overhead: one 8-Kbyte block reading 80 bytes at a time will require 102 system calls.

The number of disk I/O operations to and from cache is reported by `sar -b` as `breads` and `bwrts`.

The following displays partial output of the `sar -b` command:

```
# sar -b
```

```
SunOS rafael 5.7 Generic sun4u      06/22/99

00:00:01 bread/s lread/s %rcache bwrit/s lwrit/s %wcache pread/s pwrit/s
01:00:01      0      1     100      0      1     74      0      0
02:00:02      0      1     100      0      1     74      0      0
...
...
15:00:01      0      1      93      0      1      56      0      0
Average       0      1      95      0      1      67      0      0
#
```

Some of the fields in this output are:

- `bread/s` – Average number of reads per second submitted to the buffer cache from the disk.
- `bwrit/s` – Average number of physical blocks (512 blocks) written from the buffer cache to disk per second.



Working With the `segmap` Cache

- You can bypass the cache by using direct I/O or `mmap`.
 - I/O goes directly to and from a disk.
 - There are restrictions.
- You can make the cache write-through using `O_SYNC` and `O_DSYNC` on the `open` call.
- You can continue to execute during I/O operations using async I/O.
 - This allows non-blocking application I/O.

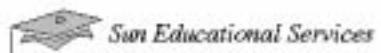
Working With the `segmap` Cache

Because of the overhead involved in accessing the `segmap` cache, the system has provided several mechanisms to bypass it.

Remember, the `segmap` cache is write back, not write through. The program will be told that I/O is complete, even though it has not started. When an application must wait until the data is on disk before continuing, it can open the file with the `O_SYNC` or `O_DSYNC` flags.

`O_DSYNC` requests a data synchronous operation; the data must be on the drive before the request is signalled complete. `O_SYNC` requests that metadata updates be complete, and performs the `O_DSYNC` processing. The use of these flags can significantly slow the application and generate extra I/O since write cancellation cannot be used, even though its use is required by an application (such as a log file).

Asynchronous I/O allows the program to continue running while the I/O is done. Standard UNIX I/O is synchronous to the application programmer.



File System Performance Statistics

From `sar -a, -g, and -v`:

- `namei/s` – DNLC pathname component requests
- `iget/s` – DNLC cache misses
- `dirbk/s` – Directory block reads per second
- `ufs_ipf` – Flushed inodes: percentage of (closed) UFS inodes flushed from the cache which had reusable file pages still in memory
- `file_sz` – Total number of open files in the system
- `inode_sz` – Current percentage of inode cache in use

File System Performance Statistics

`sar` options `-a`, `-g`, and `-v` provide data relevant to file system caching efficiency and the cost of cache misses.

Any nonzero values for `%ufs_ipf`, as reported by `sar -g`, indicate that the inode cache is too small for the current workload.

The DNLC hit rate can be calculated by using $(1 - (\text{iget} \div \text{namei})) \times 100$.

```
# sar -a

SunOS rafael 5.7 Generic sun4u      06/23/99

00:00:01  igure/s namei/s dirbk/s
01:00:00      0      2      0
02:00:00      0      2      0
...
...
09:00:01      0      3      2

Average      0      2      1
#
```

Some of the numbers reported are:

- `igure/s` – The number of requests made for inodes that were not in the DNLC.
- `namei/s` – The number of file system path searches per second. If `namei` does not find a directory name in the DNLC, it calls `igure` to get the inode for either a file or directory. Most `igure` calls are the result of DNLC misses.
- `dirbk/s` – The number of directory block reads issued per second.

```
# sar -g
```

```
SunOS rafael 5.7 Generic sun4u      06/23/99

00:00:01 pgout/s ppgout/s pgfree/s pgscan/s %ufs_ipf
01:00:00    0.01     0.02     0.05     0.62     0.00
02:00:00    0.00     0.00     0.00     0.03     0.00
...
...
09:00:01    2.44     5.54    12.68    42.08     0.00
Average     0.19     0.49     1.03     4.16     0.00
#
```

The one field that is relative to file system caching in this output is:

- %ufs_ipf – The percentage of UFS inodes taken off the free list by iget that had reusable pages associated with them. These pages are flushed and cannot be reclaimed by processes. Thus, this is the percentage of iget calls with page flushes. A high value indicates that the free list of inodes is page bound and the number of UFS inodes may need to be increased.

```
# sar -v
```

```
SunOS rafael 5.7 Generic sun4u      06/23/99

00:00:01 proc-sz      ov  inod-sz      ov  file-sz      ov  lock-sz
01:00:00    79/922      0  1821/4236    0  616/616      0   0/0
02:00:00    79/922      0  1821/4236    0  616/616      0   0/0
...
...
09:20:03    75/922      0  1911/4236    0  611/611      0   0/0
#
```

Included in this output are:

- inod-sz – The total number of inodes in memory versus the maximum number of inodes allocated in the kernel. This is not a strict high-water mark; it can overflow.
- file-sz – The number of entries and the size of the open system file table.



fsflush

- Remember that the `segmap` cache is a write-back cache.
- Data could stay there unwritten indefinitely, which is not good in case of a system crash.
- `fsflush` writes modified data back to disk on a regular basis for UFS.
- There are two tuning parameters:
 - `tune_t_fsflushr` – How often to wake up
 - `autoup` – How long a cycle should take
- `autoup` is the longest that unmodified data will remain unwritten.

fsflush

Since main memory is treated as a write-back cache, and UFS treats the `segmap` cache the same way, this implies that modified file data could remain in memory indefinitely.

This is certainly not acceptable; in the event of a system crash the amount and identity of lost data would never be known, and complete file system recovery from backups might be required. UFS tries to balance the performance gained by write-back caching with the need to prevent data loss. The result is the `fsflush` daemon.

`fsflush` wakes up every `tune_t_fsflushr` seconds (default 5) and searches for modified file system pages. It must complete an entire cycle through main storage in `autoup` seconds (default 30). This means that every awakening `fsflush` must check 5/30 or 1/6 of main memory. This might cause I/O spikes in large systems, so `tune_t_fsflushr` can be made smaller to even out the I/O load.

In systems with a large amount of physical memory, you may want to increase the value of *autooup*, along with *tune_t_fsflushr*. This will reduce the amount of CPU time used by *fsflush* by increasing its cycle time. Be aware that the trade-off is more potential data loss if a system failure occurs.



Direct I/O

- Allows I/O to bypass the `segmap` cache
- Is new with the Solaris 2.6 OS
- Works best when files are much larger than memory or data is already buffered
- Allows regular file system use in place of raw partitions
- Is enabled by:
 - Application use of `directio` call
 - `forcedirectio` mount option
- Has restrictions on its use

Direct I/O

Direct I/O is application I/O that does not pass through the `segmap` cache. Since all application I/O passes through the `segmap` cache by default, use of the direct I/O special management is required. The direct I/O facility is new with the Solaris 2.6 operating system.

Direct I/O can be requested in two ways: per file, by the application using the `directio` library call, and as the default for an entire partition using the `forcedirectio` mount option.

Using direct I/O on an entire partition allows the use of a UFS file system for data that would otherwise require raw partitions, such as database table spaces.

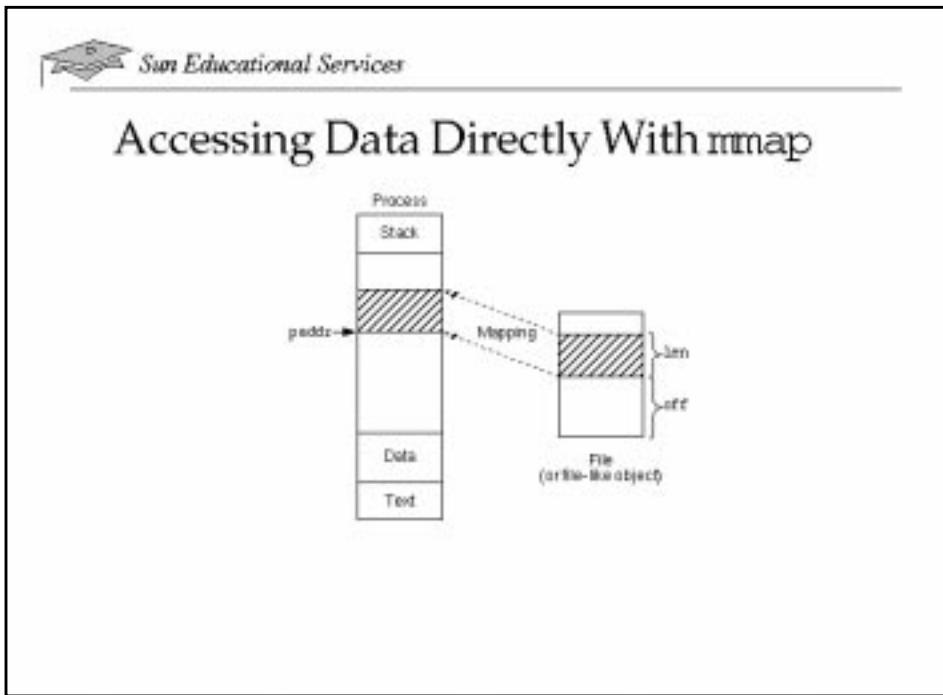
Restrictions

There are restrictions on the use of direct I/O. All of the following must be true for direct I/O to be used. If they are not met, regular I/O will be used, without notice.

- The file is not memory mapped in any other process.
- The file is not on a UFS logging file system.
- The file does not have holes (is not sparse).
- The specific read or write request is disk sector aligned (on a 512-byte seek boundary), and the request is for an integer number of sectors.

Also note:

- Direct I/O does not do read-ahead or write-behind. This must be managed by the application.
- All I/O to direct files must be complete before the application exits.
- Direct I/O cannot coexist with regular usage of the file.
- The direct I/O facility can be disabled by setting *ufs:directio_enabled* to zero.



Accessing Data Directly With mmap

Another mechanism to bypass the segmap cache is `mmap`. The `mmap` system call allows the programmer to memory map a file. This means that the file, or a portion of it (called a window), is treated as if it were a segment of the process' address space. The file thus looks like an array resident in the process' virtual memory.

The file is accessed as if it were an array, referenced by pointers or array element subscripts. Changes to the file are performed by writing to the proper memory location. `fsflush` will then cause them to be moved to disk (or the programmer might request a write).

To control the system's management of the `mmap` segment, or any virtual memory segment, the programmer can use the `madvise` call.



Using madvise

It is used to control more precisely an application's use of main storage. It can be used for most virtual memory areas.

- **MADV_NORMAL** – Normal (read ahead only) access.
- **MADV_SEQUENTIAL** – Sequential access (read ahead and free behind).
- **MADV_RANDOM** – Random access; no read ahead.
- **MADV_WILLNEED** – Virtual memory the application intends to use soon. The data will be read in by read ahead.
- **MADV_DONTNEED** – Unneeded virtual memory. Any associated physical memory will be freed.

Using madvise

madvise is used to specify how the application intends to reference pages of virtual memory. The program can specify:

- **MADV_NORMAL** – Normal access (perform read ahead).
- **MADV_RANDOM** – Random access; no read ahead.
- **MADV_SEQUENTIAL** – Sequential access (use read ahead and free behind).
- **MADV_WILLNEED** – A range of virtual addresses that the application intends to reference. These pages will be asynchronously faulted into memory (read ahead).
- **MADV_DONTNEED** – A range of virtual addresses that are no longer needed by the application. The kernel will free the system resources associated with these addresses. Any modified data will be written to disk.



File System Performance

- The workload characteristics of the application must be understood before configuring a file system.
- Data-intensive applications access large files without creating or deleting files.
- Attribute-intensive applications create and delete many small files.
- Decision support or data warehousing has sequential access I/O patterns.
- Transaction processing has random access I/O patterns.

File System Performance

File system performance is heavily dependent on the nature of the application generating the load. Before configuring a file system, you need to understand the workload characteristics that are going to use the file system.

Data-Intensive or Attribute-Intensive Applications

A data-intensive application accesses very large files. Large amounts of data are moved around without creating or deleting many files. An example of a data-intensive workload is an image processing application or a scientific batch program that creates Gbyte-size files.

An attribute-intensive application accesses many small files. An application which creates and deletes many small files and only reads and writes small amounts of data in each would be considered attribute intensive. Perhaps the best example is the software development shop.

Sequential or Random Accesses

The access pattern of an application also needs to be considered when planning an I/O configuration. Applications read and write through a file either sequentially or in a random order. Sequential workloads are easiest to tune because I/Os can be grouped before being issued to the storage device.

Online transaction processing (OLTP) is characterized by small I/O size (2 Kbytes to 16 Kbytes) and is predominantly random access. Decision support systems (DSS) are characterized by large I/O size (64 Kbytes to 1 Mbyte) and are predominately sequential access.

High performance computing (HPC) is characterized by large I/O size (1 Mbyte to 16 Mbytes) and is predominantly sequential access. Internet Service Providers (ISP) are characterized by small I/O size (1 Kbyte to 128 Kbytes) and are predominately random access.

NFS Version 2 is characterized by 8-Kbyte synchronous writes. It should be avoided whenever NFS Version 3 is available. NFS Version 3 is highly flexible and may be tailored to the needs of your application. Tuning is based on the data-intensive versus the attribute-intensive disposition of the workload.

Time share is characterized by being nearly impossible to characterize. Determine which activities share the system and then configure the most balanced configuration possible.



Cylinder Groups

- The size of the cylinder group is dependent on the type application running on the file system.
- The goal is to keep files in a single cylinder group.
 - The default of 16 cylinders per group is sufficient for attribute-intensive applications.
 - You should consider increasing number of cylinders per group to 32 (maximum) for data-intensive applications.
- The size of a cylinder group can be computed:

Number of bytes in a cg = (512 bytes/sector) x (nr. of sectors per track) x (nr. of tracks per cylinder) x (16 cylinders/cylinder group)

Cylinder Groups

Using newfs in a default way causes the system to allocate 16 cylinders per cylinder group. Recall that a cylinder is a collection of all tracks through the disk platter. Each track is composed of sectors. Each sector is 512 bytes. Every disk will have a different number of sectors per track. Given this information from the disk vendor (or from the mkfs -m command), the size of a cylinder group (cg) may be computed as follows:

Number of bytes in a cg = (512 bytes/sector) x (number of sectors per track) x (number of tracks per cylinder) x (16 cylinders/cylinder group)

For attribute-intensive file systems, this is probably a sizeable number of cylinders per cylinder group, since it is optimized for that purpose.

For data-intensive file systems, this value may be too small. If the average file size is 15 Mbytes, consider increasing the number of cylinders per cylinder group, based on the previous computation. Remember, the file system tries to keep files in a single cylinder group, so make sure that the cylinder groups are big enough.

Consider the following example:

Number of sectors per track = 224

Number of tracks per cylinder = 7

Using this computation:

Number of bytes in a cg = $512 \times 224 \times 7 \times 16 = 12,845,056$

In this example, a file system with an average file size of 15 Mbytes would benefit by increasing the number of cylinders per cylinder group to 32 (32 is the maximum).

```
# newfs -c 32 /dev/rdsk/xxx
```



Sequential Access Workloads

- This typically occurs when moving large amounts of data.
 - The truss command can be used to investigate the nature of an application.
 - Absence of lseek system calls between each read indicates application is reading sequentially.
 - The first read system call requires a physical disk read.
 - The first file system block, 8-Kbytes is read into memory.
 - Subsequent read calls get data directly from memory.
 - A read-ahead algorithm improves the performance of sequential reads.

Sequential Access Workloads

Sequential workloads read or write sequential file blocks. Sequential I/O typically occurs when moving large amounts of data, such as copying a file, or reading or writing a large portion of a file.

The `truss` command can be used to investigate the nature of an application's access pattern by looking at the `read`, `write`, and `lseek` system calls.

```
# truss -p 20432
```

By tracing the system calls generated by an application with a process ID of 20432, you can see that the application is reading 512-byte blocks. The absence of `lseek` system calls between each `read` indicates that the application is reading sequentially.

When the application requests the first read, the file system reads in the first file system block, 8 Kbytes, for the file. This operation requires a physical disk read, which takes a few milliseconds. The second 512-byte read will read the next 512 bytes from the 8-Kbyte file system block that is still in memory. This only takes a few hundred microseconds.

This is a major benefit to the performance of the application. Reading the physical disk in 8-Kbyte blocks means that the application only needs to wait for a disk I/O every 16 reads rather than every read, reducing the amount of time spent waiting for I/O.

Waiting for a disk I/O every sixteen 512-byte reads is still terribly inefficient. Most file systems implement a read-ahead algorithm in which the file system can initiate a read for the next few blocks as it reads the current block. Because the access pattern is repeating, the file system can predict that it is very likely the next block will be read, given the sequential order of the reads.



UFS File System Read Ahead

- The UFS file system determines when to implement read ahead.
- Several criteria must be met before engaging read ahead.
 - Memory-mapped files do not use UFS read ahead.
 - Cluster size (*maxcontig* parameter) determines number of blocks to read ahead.
 - *maxcontig* defaults to 16 8-Kbyte blocks (128-Kbytes) beginning with the Solaris 2.6 OS.
 - Default value is often too low to allow optimal read rates.
 - *maxcontig* can be viewed by `mkfs -m` command

UFS File System Read Ahead

The UFS file system determines when to implement read ahead by keeping track of the last read operation. If the last read and the current read are sequential, a read ahead of the next sequential series of file system blocks is initiated.

The following criteria must be met to engage UFS read ahead: the last file system read must be sequential with the current one, there must be only one concurrent reader of the file (reads from other processes will break the sequential access pattern for the file), the blocks of the file being read must be sequentially layered out on the disk, and the file must be being read or written via the read and write system calls; memory-mapped files do not use the UFS read ahead.

The UFS file system uses the cluster size (*maxcontig* parameter) to determine the number of blocks that are read ahead. This defaults to seven 8-Kbyte blocks (56 Kbytes) in Solaris versions up to the Solaris 2.6 OS. In Solaris 2.6, the default changed to the maximum size transfer supported by the underlying device, which defaults to sixteen 8-Kbyte blocks (128 Kbytes) on most storage devices.

The default values for read ahead are often too low and should be increased to allow optimal read rates. The cluster size should be set very large for high-performance sequential access to take advantage of modern I/O systems.

The behavior of the 512-byte read example is displayed by `iostat`.

```
# iostat -x 5
```

device	r/s	w/s	kr/s	kw/s	wait	actv	svc_t	%w	%b
fd0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0	0
sd6	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0	0
ssd11	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0	0
ssd12	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0	0
ssd13	49.0	0.0	6272.0	0.0	0.0	3.7	73.7	0	93

```
#
```

Issuing 49 read operations per second to disk `ssd113` and averaging 6,272 Kbytes per second from the disk results in the average transfer size of 128 Kbytes. This confirms that the default 128-Kbyte cluster size is grouping the 512-byte read requests into 128-Kbyte groups.

The cluster size of a UFS file system (`maxcontig` parameter) can be displayed by using the command, `mkfs -m`.

```
# mkfs -m /dev/rdsck/c0t0d0s0
```

```
mkfs -F ufs -o nsect=80,ntrack=19,bsize=8192,fragsize=1024,
cgszie=32,free=4,rps=90,nbpi=4136,opt=t,apc=0,gap=0,nrpos=8,
maxcontig=16 /dev/rdsck/c0t0d0s0 3366800
#
```

For this file system, you can see that the cluster size (`maxcontig`) is 16 blocks of 8,192 bytes, or 128 Kbytes.



Cluster Sizes for RAID Volumes

- Cluster sizes larger than 128 Kbytes require the *maxphys* parameter to be set in */etc/system*.
- The guidelines for cluster sizes on RAID devices are as follows:
 - RAID -0 striping: Cluster size is equal to number of stripe members multiplied by the stripe size.
 - RAID -1 mirroring: Cluster size is the same as for a single disk.
 - RAID -0+1 striping + mirroring: Cluster size is equal to number of stripe members per mirror multiplied by the stripe size.

Setting Cluster Sizes for RAID Volumes

SCSI drivers in the Solaris environment limit the maximum size of an SCSI transfer to 128 Kbytes by default. Even if the file system is configured to issue 512-Kbyte requests, the SCSI drivers will break the requests into smaller 128-Kbyte chunks.

The same limit applies to volume managers like Solstice DiskSuite™ and Veritas Volume Manager (VxVM). Whenever you use a device that requires larger cluster sizes, you need to set the SCSI and volume manager parameters in the */etc/system* configuration file to allow bigger transfers. The following changes in the */etc/system* file provide the necessary configuration for larger cluster sizes.

To allow larger SCSI I/O transfers (parameter is bytes) use:

```
set maxphys = 1048576
```

To allow larger DiskSuite I/O transfers (parameter is bytes) use:

```
set md_maxphys = 1048576
```

To allow larger VxVM I/O transfers (parameter is 512 byte units) use:

```
set vxio:vol_maxio = 2048
```

Since RAID devices and volumes often consist of several physical devices arranged as one larger volume, you need to pay attention to what happens to these large I/O requests when they arrive at the RAID volume.

For example, a simple RAID level 0 stripe can be constructed from seven disks in a Sun A5200 storage subsystem. I/Os are then interlaced across each of the seven devices according to the interlace size or stripe size. By interlacing I/O operations across separate devices, you can potentially break up the I/O that comes from the file system into several requests that can occur in parallel.

With seven separate disk devices in the RAID volume, you have the ability to perform seven I/O operations in parallel. Ideally, you should also have the file system issue a request that initiates I/Os on all seven devices at once. To split the I/O into exactly seven components when it is broken up, you must initiate an I/O the size of the entire stripe width of the RAID volume. This requires you to issue I/Os that are 128 Kbytes X 7, or 896 Kbytes each. In order to do this, you must set the cluster size to 896 Kbytes.

RAID-5 is similar but you only have an effective space of $n - 1$ devices because of the parity stripe. Therefore, an eight-way RAID-5 stripe will have the same stripe width and cluster size as a seven-way RAID-0 stripe.

The guidelines for cluster sizes on RAID devices are as follows:

- RAID-0 striping: Cluster size is equal to the number of stripe members multiplied by the stripe size.
- RAID-1 mirroring: Cluster size is the same as for a single disk.
- RAID-0+1 striping and mirroring: Cluster size is equal to the number of stripe members per mirror multiplied by the stripe size.



Commands to Set Cluster Size

- Use `newfs -C #` to set cluster size (where # is the number of 8-Kbyte blocks)
 - `newfs -C 112 /dev/rdsk/xxx` creates a file system with a cluster size of 896 Kbytes.
- Use `tunefs -a #` to modify the cluster size for an existing file system (where # is the number of 8-Kbyte blocks)
 - `tunefs -a 112 /dev/rdsk/xxx` changes the cluster size to 896 Kbytes.
- Remember, for RAID devices, `maxcontig` should be set so that the cluster size is the same as the stripe width.

Commands to Set Cluster Size

The cluster size can be set at the time the file system is created using the `newfs` command. Cluster size can be modified after the file system is created by using the `tunefs` command. To create a UFS file system with an 896-Kbyte cluster size command, issue the following command:

```
# newfs -C 112 /dev/rdsk/xxx
```

To modify the cluster size after the file system has been created using the `tunefs -a` command, use:

```
# tunefs -a 112 /dev/rdsk/xxx
```

Note – It is important to note that the UFS read ahead algorithms do not differentiate between multiple readers; thus, two processes reading the same file will break the read ahead algorithms.

Tuning for Sequential I/O

Tuning for sequential I/O involves maximizing the size of the I/Os from the file system to disk. This also reduces the amount of CPU required and the number of disk I/Os.

To maximize the amount of I/O to a file system, the file system needs to be configured to write as large as possible I/Os out to disk. The *maxcontig* parameter is also used to cluster blocks together for I/Os. *maxcontig* should be set so that the cluster size is the same as the stripe width in a RAID configuration.

In a RAID configuration with four disks in a stripe and a stripe size of 16 Kbytes, the stripe width is 64 Kbytes. To optimize I/O in this case, set *maxcontig* to 8 (64 Kbytes ÷ 8 Kbytes). To configure this when creating the file system, use the command:

```
# newfs -C 8 /dev/rdsk/xxx
```

To modify an existing file system, use the command:

```
# tunefs -a 8 /dev/rdsk/xxx
```



File System Write Behind

- Delayed asynchronous writes is the UNIX file system's default method for writing data blocks.
- This allows the application to continue to process without having to wait for each I/O.
- This also allows the operating system to delay the writes long enough to group together adjacent writes.
- The *maxcontig* parameter controls how many writes are grouped together before a physical write is performed.
- Guidelines used for read ahead should also be applied to write behind.

File System Write Behind

UNIX employs an efficient method to process writes. Rather than writing each I/O synchronously, UNIX passes the writes to the operating system. This allows the application to continue processing. This method of delayed asynchronous writes is the UNIX file system's default method for writing data blocks. Synchronous writes are used only when a special file option is set.

Delayed asynchronous writes allow the application to continue to process without having to wait for each I/O; it also allows the operating system to delay writes long enough to group together adjacent writes. When writing sequentially, this allows a few large writes to be issued which is far more efficient than issuing several small writes.

The UFS file system uses the same cluster size parameter, *maxcontig*, to control how many writes are grouped together before a physical write is performed. The guidelines used for read ahead should also be applied to write behind. Again, if a RAID device is used, care should be taken to align the cluster size to the stripe size of the underlying device.

The following example shows the I/O statistics for writes generated by the `mkfile` command, which issues sequential 8-Kbyte writes:

```
# mkfile 500m testfile&
# iostat -x 5

device    r/s   w/s   kr/s   kw/s  wait  actv  svc_t  %w   %b
sd3       0.0   0.0   0.0    0.0   0.0   0.0   0.0    0   0
ssd49     0.0   0.0   0.0    0.0   0.0   0.0   0.0    0   0
ssd50     0.0   0.0   0.0    0.0   0.0   0.0   0.0    0   0
ssd64     0.0  39.8   0.0  5097.4   0.0  39.5  924.0   0 100
#
```

The `iostat` command shows 39.8 write operations per second to the disk `ssd64`, averaging 5,097.4 Kbytes/sec from the disk. This results in an average transfer size of 128 Kbytes.

Changing the cluster size to 1 Mbyte, or 1,024 Kbytes requires setting `maxcontig` to 128, which represents one hundred and twenty-eight 8-Kbyte blocks, or 1 Mbyte. `maxphys` must also be increased as described earlier.

```
# tunefs -a 128 /ufs
# mkfile 500m testfile&

# iostat -x 5

device    r/s   w/s   kr/s   kw/s  wait  actv  svc_t  %w   %b
sd3       0.0   0.0   0.0    0.0   0.0   0.0   0.0    0   0
ssd49     0.0   0.0   0.0    0.0   0.0   0.0   0.0    0   0
ssd50     0.0   0.0   0.0    0.0   0.0   0.0   0.0    0   0
ssd64     0.2   6.0   1.0  6146.0   0.0   5.5  804.4   0   99
#
```

Now `iostat` shows 6.0 write operations per second, averaging 6,146 Kbytes/sec from the disk. Dividing the transfer rate by the number of I/Os per second results in an average transfer size of 1,024 Kbytes.

Note – The UFS clustering algorithm will only work properly if one process or thread writes to the file at a time. If more than one process or thread writes to the same file concurrently, the delayed write algorithm in UFS will begin breaking up the writes into random sizes.



UFS Write Throttle

- Prevents the file system from flooding the system memory with outstanding I/O requests
- Uses three parameters to control the throttling mechanism:
 - When *ufs_WRITE* is set to 1 (default), the UFS write throttle is enabled.
 - Writes are suspended when the number of outstanding writes is larger than *ufs_HW*.
 - Writes are resumed when the number of writes falls below *ufs_LW*.
- Uses default parameters to prevent full sequential write performance of most disks

UFS Write Throttle

The UFS file system limits the amount of outstanding I/O (waiting to complete) to 384 Kbytes per file. If a process has a file that reaches the limit, the *entire process* is suspended until the resume limit has been reached. This prevents the file system from flooding the system memory with outstanding I/O requests.

The default parameters for the UFS write throttle prevent you from using the full sequential write performance of most disks and storage systems. If you ever have trouble getting a disk, stripe, or RAID controller to show up as 100 percent busy when writing sequentially, the UFS write throttle is the likely cause.

Two parameters control the write throttle: the high-water mark, *ufs_HW*, and the low-water mark, *ufs_LW*. The UFS file system suspends writing when the amount of outstanding writes grows larger than the number of bytes in *ufs_HW*; it resumes writing when the amount of writes falls below *ufs_LW*.

A third parameter, *ufs_WRITEs*, determines whether or not file system watermarks are enabled. File system watermarks are enabled by default because the value for *ufs_WRITEs* is 1. *ufs_WRITEs* should not be set to 0 (disabled) because this could lead to the entire system memory being consumed with I/O requests. *ufs_HW* and *ufs_LW* have meaning only if *ufs_WRITEs* is not equal to zero.

ufs_HW and *ufs_LW* should be changed together to avoid needless churning when processes awake and find that they either cannot issue a write (the parameters are too close) or have waited longer than necessary (the parameters are too far apart).

The write throttle can be permanently set in /etc/system. Set the write throttle high-water mark to 1/64 of the total memory size and the low water mark to 1/128 of the total memory size. For a 1-Gbyte machine, you would set *ufs_HW* to 16,777,216 and *ufs_LW* to 8,388,608.

```
set ufs:ufs_HW=16777216
```

```
set ufs:ufs_LW=8388608
```



Random Access Workloads

- Typically this occurs when small amounts of data are being moved.
- OLTP, NFS servers with many small files, and ISP servers typically have random access workloads.
- You can use the `truss` command to investigate the nature of an application.
 - The presence of a `lseek` system call between each `read` indicates the application is reading randomly.
 - The third argument of `read` system call shows block size.

Random Access Workloads

The `truss` command can be used to investigate the nature of an application's access pattern. In the following output, the `truss` command is used to trace the system calls generated by an application whose process ID is 19231:

```
#truss -p 19231
```

```
lseek(3, 0x0D780000, SEEK_SET)          = 0x0D780000
read(3, 0xFFBDF5B0, 8192)                = 0
lseek(3, 0xA6D0000, SEEK_SET)          = 0xA6D0000
read(3, 0xFFBDF5B0, 8192)                = 0
lseek(3, 0xFA58000, SEEK_SET)          = 0xFA58000
read(3, 0xFFBDF5B0, 8192)                = 0
lseek(3, 0xF79E000, SEEK_SET)          = 0xF79E000
read(3, 0xFFBDF5B0, 8192)                = 0
lseek(3, 0x080E4000, SEEK_SET)          = 0x080E4000
read(3, 0xFFBDF5B0, 8192)                = 0
lseek(3, 0x024D4000, SEEK_SET)          = 0x024D4000
```

Use the arguments from the `read` and `lseek` system calls to determine the size of each I/O and the seek offset at which each `read` is performed.

The `lseek` system call shows the offset within the file in hexadecimal. In this example, the first two seeks are to offset 0x0D780000 and 0xA6D0000, respectively. These two addresses appear to be random, and further inspection of the remaining offsets shows that the `reads` are completely random.

Look at the argument to the `read` system call to see the size of each `read`, which is the third argument. In this example, every `read` is exactly 8,192 bytes, or 8 Kbytes. Thus you can see that the seek pattern is completely random, and that the file is being read in 8-Kbyte blocks.



Tuning for Random I/O

- Try to match the I/O size and the file system block size
- Choose an appropriately large file system cache
- Disable prefetch and read ahead, or limit read ahead to the size of each I/O
- Disable file system caching when the application does its own caching
- When configuring a RAID device for random access workloads, use many small disks
- Set the stripe size of a RAID device small to reduce latency and the number of physical blocks transferred

Tuning for Random I/O

There are several factors that should be considered when configuring a file system for random I/O:

- Try to match the I/O size and the file system block size.
- Choose an appropriately large file system cache.
- Disable prefetch and read ahead, or limit read ahead to the size of each I/O.
- Disable file system caching when the application does its own caching (for example, in databases).

Try to match the file system block size to a multiple of the I/O size for workloads that include a large proportion of writes. A write to a file system that is not a multiple of the block size requires the old block to be read, the new contents to be updated, and the whole block to be written out again. Applications that do odd-sized writes should be modified to pad each record to the nearest possible block size multiple to eliminate the read-modify-write cycle. The `-b` option for the `newfs` command sets the block size in bytes.

Random I/O workload often accesses data in very small blocks (2 Kbytes to 8 Kbytes), and each I/O to and from the storage device requires a seek and an I/O because only one file system block is read at a time. Each disk I/O takes a few milliseconds, and while the I/O is occurring, the application must wait for it to complete. This can represent a large proportion of the application's response time. Caching file system blocks into memory can make a big difference to an application's performance because you can avoid many of those expensive and slow I/Os.

For example, consider a database that does three reads from a storage device to retrieve a customer record from disk. If the database takes 500 microseconds of CPU time to retrieve the record, and spends 3 to 5 milliseconds to read the data from disk, it spends a total of 15.5 milliseconds to retrieve the record; 97 percent of that time is spent waiting for disk reads.

You can dramatically reduce the amount of time spent waiting for I/Os by using memory to cache previously read disk blocks. If that disk block is needed again, it is retrieved from memory. This eliminates the need to go to the storage device again.

Setting the cluster size, *maxcontig*, to 1 will reduce the size of the I/Os from the file system to disk. Use `newfs -C 1 /dev/rdsk/xxx` when creating the file system and `tunefs -a 1 /dev/rdsk/xxx` when modifying an existing file system.

When configuring a RAID device for random access workloads, use many small disks. Setting the stripe size of a RAID device to a small number helps reduce latency and the amount of transferred physical blocks required to read the small blocks of random access workloads.



Post-Creation Tuning Parameters

Parameters on `tunefs` (and `newfs/mkfs`) are:

- Optimization for time or space
 - `time` – Allocate a new block for each fragment section
 - `space` – Allocate fragment's best-fit
- `minfree` – Reserve space to cut quadratic rehash short
 - Formula is: $(64 \text{ Mbytes} + \text{file system size}) \times 100$
 - Default is 10 percent
- `maxbpg` - Help keep large files more contiguous

Post-Creation Tuning Parameters

Once a partition has been created, many of its characteristics are fixed until it is re-created. There are several post-creation strategies that can be used to affect the performance of the partition using `tunefs`.

Current partition configuration settings can be determined by using `fstyp -v`.

Fragment Optimization

As discussed earlier, if a file system is optimized for time and if there is not enough room in the last block of the file to occupy another fragment, when a file needs to grow, the kernel selects an empty block and copies the file's fragments from the old block to the new block.

If a file system is optimized for space and there is not enough room in the last block to occupy another fragment, when a file needs to grow, the kernel looks for a block with the number of free fragments equal to exactly what the file needs and copies the fragments to this new block. In other words, it uses a best-fit policy.

minfree

`minfree` is used to force a percentage of blocks in the file system to always remain unallocated. When a file system is almost full, performance of the allocation routines for data blocks and `inodes` drops due to the quadratic rehash process.

In file systems created before SunOS 5.6, `minfree` is most likely to be large. Using the algorithm given earlier in the quadratic rehash discussion, you can adjust the value of `minfree` in your existing partitions.

In a file system that is read-only, `minfree` should be dropped from the default 10 percent to 0 percent since data block and `inode` allocation are not a concern.

maxbpg

The UFS file allocation system will only allow a certain number of blocks from a file in each cylinder group. The default is 16 MBytes, which is usually larger than the default size of a cylinder group. To ensure that a large file can use the entire cylinder group (keeping the file more contiguous), increase this limit. This limit is set with the `tunefs -e` parameter, and cannot be set with `newfs`. A recommended value that should work for all cylinder group sizes is 4,000,000.

Do not change this value if performance of small files in the file system is important.



Application I/O Performance Statistics

From `sar -b` you can get segmap and buffer cache behavior.

- `bread/s`, `bwrit/s` – segmap cache I/O operations.
- `lread/s`, `lwrit/s` – Buffer cache I/O operations.
- `%rcache`, `%wcache` – segmap cache hit ratio.
- `pread/s`, `pwrite/s` – `physio()` transfers. `physio()` is used when accessing raw devices.

Application I/O Performance Statistics

`sar -b` reports on the segmap and metadata caches. The overhead image above shows the available information.

```
# sar -b
```

```
SunOS rafael 5.7 Generic sun4u      06/23/99
```

	<code>bread/s</code>	<code>lread/s</code>	<code>%rcache</code>	<code>bwrit/s</code>	<code>lwrit/s</code>	<code>%wcache</code>	<code>pread/s</code>	<code>pwrite/s</code>
00:00:01								
01:00:00	0	1	100	0	1	74	0	0
02:00:00	0	0	100	0	1	74	0	0
...								
...								
14:40:00	0	1	100	0	1	72	0	0
Average	0	1	99	0	1	69	0	0
#								

Fields in this output include:

- `bread/s` – Average number of reads per second submitted to the buffer cache from the disk.

- `bwrit/s` – Average number of physical blocks (512 blocks) written from the buffer cache to disk per second.
- `lread/s` – Average number of logical reads per second from the buffer cache.
- `lwrit/s` – Average number of logical writes to the buffer cache per second.
- `%rcache` – Fraction of logical reads found in the buffer cache (100 percent minus the ratio of `bread/s` to `lread/s`).
- `%wcache` – Fraction of logical writes found in the buffer cache (100 percent minus the ratio of `bwrit/s` to `lwrit/s`).
- `pread/s` – Average number of physical reads, per second, using character device interfaces.
- `pwrite/s` – Average number of physical write requests, per second, using character device interfaces.

Check Your Progress

Before continuing on to the next module, check that you are able to accomplish or answer the following:

- Describe the vnode interface layer to the file system
- Explain the layout of a UFS partition
- Describe the contents of an `inode` and a directory entry
- Describe the function of the superblock and cylinder group blocks
- Understand how allocation is performed in the UFS
- List the UFS caches and how to tune them
- Describe how to use the other UFS tuning parameters

Think Beyond

Consider these questions:

- What will the effect on UFS be if there is not enough main storage?
- Why does the UltraSPARC not support the 4-Kbyte UFS block size?
- When would the use of a logging file system be critical to performance?
- What are the most expensive UFS operations? Why can not they be avoided?

Objectives

Upon completion of this lab, you should be able to:

- Determine how a partition was created
- See the performance benefit of the DNLC
- Determine how many inodes a partition needs
- See the benefits of using `mmap` and `madvise`

Tasks

File Systems

Complete the following steps:

1. Look at the newfs/mkfs parameters used to create your root partition and record the following information:

```
fstyp -v /dev/rdsck/cXtXdXsX | more
```

size_____

ncg_____

minfree_____

maxcontig_____

optim_____

nifree_____

ipg_____

What newfs command was used to create this partition?

Using the values for nifree and ipg, calculate how many allocated or used inodes are present in the partition. Add 50 percent to this number as a reasonable approximation of how many might be needed. Check your answer with df -Fufs -o i.

At 128 bytes per inode, how much space is wasted with extra inodes? What percentage of the partition space is this?

What value for `-i` should be used with `newfs` if the partition is re-created?

What should the value for `minfree` be for this partition?

DirectoryName Lookup Cache

Use the files in the `lab9` directory for these exercises.

1. Examine the effects of running a system with a very small `dnlc` and `inode` cache. `ncsize` is the size of the `dnlc`, and `ufs_ninode` is the size of the `inode` cache. Type:

`./dnlc.sh`

Record the size of both caches.

`ncsize` _____

`ufs_ninode` _____

2. Type:

`vmstat -s`

The `toolong` field represents directory names that could not be stored in the `dnlc` because they were too long (greater than 31 characters). `total name lookups` represents the number of lookups that have been made for directory names in the `dnlc`. Record the `total name lookups` (cache hits ##) and `toolong` fields.

`total name lookups` _____

`cache hits` _____

`toolong` _____

Note – The `vmstat toolong` field will not be present if you are running the Solaris 7 OS.

3. Edit `/etc/system`. Add the following lines:

```
set ncsiz e = 50  
set ufs_ninode = 50
```

4. Reboot your system.
5. After it has rebooted, type:

vmstat -s

Record the total name lookups (cache hits ##) and toolong fields.

total name lookups_____

cache hits _____

toolong _____

6. Type:

./tester_1 50

This test takes a minute or so to run.

7. When the test has completed, type:

vmstat -s

Record the results. What has happened to the dnlc hit rate? Why? (Hint – It could one of two reasons—the cache was too small or the directory names that were being referenced were too large.) What can you deduce just from the vmstat output?

total name lookups_____

cache hits _____

toolong _____



8. Type:

```
./tester_s 50
```

9. When the test has completed, type:

```
vmstat -s
```

What changes do you note from the previous runs of `tester`? Did the cache hit rate rise? Why or why not?

total name lookups _____

cache hits _____

toolong _____

10. Type:

```
sar -v 1
```

Note that the size of the `inode` table corresponds to the value you set for `ufs_ninode` and that the total number of `inodes` exceeds the size of the table. This is not a bug. `ufs_ninode` acts as a soft limit. The actual number of `inodes` in memory may exceed this value. However, the system attempts to keep the size below `ufs_ninode`.

11. Edit `/etc/system`. Remove the lines that were added earlier.

12. Reboot your system.

13. After the system has rebooted, type:

```
./tester_s 50
```

14. To verify an improvement in performance with respect to the DNLC, type:

```
vmstat -s
```

mmap and read/write

Use the files in the lab9 directory for these exercises.

1. Use your favorite editor to view the source for the program, map_file_f_b.c.

This program sequentially scans a file, either forward or backward. What system call is the program using to access the file?

Why is msync() called at the end of the program?

Which should take more time – scanning the file forward or scanning the file backward?

Why?

2. There is a file called test_file in your home directory. It is used by some of the programs you will be running. Use df -n to verify that this file resides in a (local) ufs file system. If it is *not* in a ufs file system, perform the following steps:

- a. Copy the files map_file_f_b, mf_f_b.sh, and test_file to a directory on a ufs file system. map_file_f_b and mf_f_b.sh are in your lab9 directory. test_file is in your home directory.

You will be looking at some statistics related to disk I/O; therefore, test_file, map_file_f_b, and mf_f_b.sh need to be on a local file system.

- b. Change (cd) to the directory containing test_file, map_file_f_b, and mf_f_b.sh.

3. Type:

mf_f_b.sh f

This script runs `map_file_f_b` several times and prints out the average execution time. The `f` argument tells `map_file_f_b` to sequentially scan the file starting at the first page and finishing on the last page.

Note the results.

System total _____ Average _____

User total _____ Average _____

Real-time total _____ Average _____

4. Type:

mf_f_b.sh b

This runs `map_file_f_b` several times, but this time scanning begins on the last page of the file and works back to the first page.

Note the results.

System total _____ Average _____

User total _____ Average _____

Real-time total _____ Average _____

Which option took a shorter amount of time? _____ Why?

5. Change (`cd`) back to the `lab9` subdirectory.

Using `madvise`

Experiment with `madvise` as follows:

1. Look at `file_advice.c`. What does this program do?
-

What options does it accept and what effect do they have on the program?

2. Copy `file_advice` to the directory containing `test_file`.
3. Change (`cd`) to the directory containing `test_file`.
4. In one window, type:

`vmstat 5`

When `file_advice` has finished accessing the mapped file, it prompts you to press Return to exit. *Do not press Return*. Instead, note the change in free memory from when the program was started until `file_advice` prompted you to press Return.

5. Type:

`file_advice n`

The `n` option tells `file_advice` to use “normal” paging advice when accessing its mappings.

Note the pagein activity. What caused this?

After `file_advice` walked through the mapping, were the pages it used freed up? (Did you see a major drop in `freemem` in the `vmstat` output?) Explain.

6. Press Return so `file_advice` can exit.
7. Type:

`file_advice s`

The `s` tells `file_advice` to use “sequential” advice when accessing its mappings.

When `file_advice` prompts you to press Return, was any of the memory that had been used to access the mapping freed? (Did you see a smaller drop in `freemem` in the `vmstat` output.) Explain.

8. Press Return so `file_advice` can exit.
9. Type:

`file_advice s`

Note the number of I/O requests that were posted to the disk `test_file` resides on (that is, `s0`).

10. Press Return so `file_advice` can exit.
11. Type:

`file_advice r`

The `r` option tells `file_advice` to use “random” advice when accessing its mappings.

Note the number of I/O requests that were posted to the disk that `test_file` resides on (that is, `s0`).

Is there a difference between the sequential case and the random case?

What might account for this difference?_____

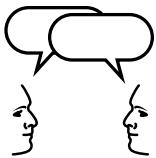
12. Press Return so `file_advice` can exit.

Objectives

Upon completion of this module, you should be able to:

- Describe the different types of network hardware
- List the performance characteristics of networks
- Describe the use of IP (internetwork protocol) trunking
- Discuss the NFS performance issues
- Discuss how to isolate network performance problems
- Determine what can be tuned

Relevance



Discussion – The following questions are relevant to understanding the content of this module:

- What can be tuned in a network?
- How important is tuning the network servers and clients?
- How do you know what to tune?

Additional Resources



Additional resources – The following references can provide additional details on the topics discussed in this module:

- Cockcroft, Adrian, and Richard Pettit. 1998. *Sun Performance and Tuning*. Palo Alto: Sun Microsystems Press/Prentice Hall.
- Wong, Brian L. 1997. *Configuration and Capacity Planning for Solaris Servers*. Palo Alto: Sun Microsystems Press/Prentice Hall.
- *SMCC NFS Server Performance Tuning Guide* in the Hardware AnswerBook on the SMCC Supplements CD-ROM.
- Solaris 7 Answerbook, *System Administration Guide*.
- Man pages for the various commands (`ndd`, `ping`, `spray`, `snoop`) and performance tools (`nfsstat`, `netstat`).



Networks

- Networks are generally packet switched buses.
- There are several places to tune:
 - The application
 - The OS
 - The hardware
 - The network itself
- You will tune differently depending on your workload.

Networks

The network system has three areas to tune, just like the I/O subsystem, although most people concentrate on just one.

A network operates like any other bus, with bandwidth and access issues. Just like any bus, speed and capacity issues play an important role in performance. This is generally not susceptible to any tuning at the individual system level, as the network infrastructure is not adjustable from OS options. Tuning the enterprise network is well beyond the scope of this course.

The other aspects of tuning the network, as in the I/O subsystem, involve making efficient use of the network. Caching where appropriate, writing cancellation opportunities, and using network protocol tuning options can greatly increase the efficiency and capacity of the network.

Each system connected to the network needs to be tuned differently for the network, depending on its workload.



Sun Educational Services

Network Bandwidth

Network Connection	Bandwidth (bits/second)
28800 baud modem	28.8 Kbytes
ISDN 1 BRI	64 Kbytes
ISDN 2 BRI	128 Kbytes
T1	1.5 Mbytes
Ethernet	10 Mbytes
T3	45 Mbytes
FastEthernet	100 Mbytes
Full Duplex FastEthernet	100+100 Mbytes
FDDI	100 Mbytes
ATM (OC3)	155+155 Mbytes
ATM (OC12)	622+622 Mbytes

Network Bandwidth

There are many different network technologies, providing a wide range of bandwidths depending on traffic needs and system capabilities.

Remember that the figures in the above overhead are bandwidths. Many of the networks are contention based, unlike the SCSI bus, and are unable to achieve practical throughput beyond 30–60 percent of these speeds.



Full Duplex 100-BaseT Ethernet

- Fast Ethernet adapters *autosense* the network characteristics.
 - It automatically determines how the network is operating.
 - Sometimes the adapter makes mistakes.
 - Speed is usually correct, duplexing is sometimes wrong.
 - You can use `ndd` to inspect and change the adapter's operation.

Full Duplex 100-BaseT Ethernet

100-base-T Ethernet (Fast Ethernet) interfaces automatically adjust their operational characteristics based on what they sense in the network they are attached to (this is called *autosensing*).

These characteristics include:

- Speed (10 Mbits or 100 Mbits per second)
- Duplexing (full or half)

It is not uncommon to have the interface sense the network incorrectly and operate with less capability than it can. Usually, it operates at the correct speed, but it might be operating in half-duplex mode instead of full duplex.

You can use the `ndd` command to check (and set) the operational characteristics of a network interface.

ndd Examples

There are several 100-BaseT full duplex parameters. Only those beginning with *adv_* should be adjusted.

```
# ndd /dev/hme instance 0 \?
... (output edited)
lp_10fdx_cap          (read only)
lp_10hdx_cap          (read only)
instance               (read and write)
lance_mode             (read and write)
ipg0                  (read and write)
... (output edited)
# ndd /dev/hme adv_100fdx_cap
0
# ndd /dev/hme 100fdx_cap
1
# ndd /dev/hme adv_10fdx_cap
0
# ndd /dev/hme 10fdx_cap
1
#
```

Forcing 100-Base T Full Duplex

To force 100-BaseT operation using ndd, set *adv_autoneg_cap* and *adv_100hdx_cap* to zero, and *adv_100_fdx_cap* to one. To set them through /etc/system, use:

```
set hme:hme_adv_autoneg_cap=0
set hme:hme_adv_100hdx_cap=0
set hme:hme_adv_100fdx_cap=1
```



IP Trunking

- Is like network striping
 - Data is striped over multiple network interfaces.
- Allows multiple interfaces to look like one
- Works only with qfe cards
 - The 100base-T quad fast Ethernet card
- Can combine two or all four interfaces
- Usually goes to an Ethernet switch
- Requires Sun Trunking™ software

IP Trunking

IP trunking is a capability provided by the 100base-T Quad Fast Ethernet (qfe) cards. It allows either two or all four interfaces to be combined as one higher speed interface, sort of like a network RAID or interleaving capability. Load balancing over the interfaces, failure recovery, and full Ethernet capabilities are provided.

Generally the IP trunking connection goes to an Ethernet switch, although there are other uses for it. It presents only one MAC (media access control) address for the aggregate connection.

To enable and configure IP trunking, you must install the Sun Trunking™ software.

Sun Trunking Software

Sun Trunking software multiplies network bandwidth in existing Gigabit Ethernet and Fast Ethernet networks. Sun Trunking software doubles the throughput of Gigabit Ethernet ports. The software also increases the throughput of Fast Ethernet networks by eight times on a single, logical port.

Sun Trunking software aggregates or "trunks" multiple network ports to create high-bandwidth network connections to alleviate bottlenecks in existing Ethernet networks. This level of network throughput will ease network bandwidth constraints in applications requiring the transport of large data files in a short period of time; such as data warehousing, data mining, database backup, digital media creation, and design automation. The technology will also be beneficial in applications that require an extremely reliable and resilient network; such as financial services.

The Sun Trunking software works in existing Gigabit Ethernet and Fast Ethernet networks and is compatible with switches from leading switch vendors including 3Com Corporation, Nortel Networks, Cabletron, Cisco, Extreme Networks and Foundry Networks. Sun Trunking software helps protect customers' current equipment investments while providing a smooth migration path to next generation networks.

Sun Trunking software allows the aggregation of multiple ports, which increases the reliability and resiliency of network connections. In the event that a single link fails, traffic can be smoothly redistributed over the remaining links. In addition, the software's load balancing feature more fully utilizes the aggregated links to provide more throughput over multiple links.



TCP Connections

Transmission Control Protocol connections are expensive.

- TCP is optimized for reliable data on long-lived connections.
- Making a connection uses a lot more CPU resources than moving data.
- Connection setup protocol involves several round-trip delays.
- Pending connections can cause "listen queue" issues.
- Each new connection goes through a "slow start" as reliability is determined.

TCP Connections

Transmission Control Protocol (TCP) connections require more system processing and resources than a User Datagram Protocol (UDP) connection. You can think of UDP connections as packet-switched connections, where TCP connections are circuit switched. This implies more overhead and processing are required for a TCP connection.

Just setting up a TCP connection requires several round-trip messages between the two participating systems, even before the actual "work" starts. On the receiving side, one connection request, taken from the "listen queue" of pending requests, is processed at a time.

In addition, since the reliability of the new connection is unknown, TCP transmits only a few messages at a time until it discovers the stability of the connection, then it begins sending more, creating more efficiency. There are TCP/IP stack tuning parameters that affect all of these areas.



TCP Connections

- TCP windows can limit high-latency, high-speed links.
- Lost or delayed data causes timeouts and retransmissions (overhead).
- Each open connection consumes memory.
- Hypertext Transfer Protocol (HTTP) persistent connections carry several operations on one connection.

TCP performs lost message recovery as part of its reliability protocols. This means that, after a message has been sent, if a response is not received within a certain amount of time, the message is assumed to have been lost and is retransmitted. If the message has been transmitted over a high latency media, such as a satellite link, messages can unnecessarily be transmitted. This leads to significant bandwidth waste, as well as unnecessary server overhead.

Also, each open TCP connection uses memory, since connection and state information must be kept.

Hypertext Transfer Protocol (HTTP) persistent connections can also add to the overhead, even though they allow several operations to be transmitted over the same connection. The volume of the connections can lead to significant resource requirements.



TCP Stack Issues

- Ignore everything if TCP throughput is less than 2 Kbytes per second.
- Warn if retransmit rate is over 15 percent; it is a problem if rate is over 25 percent.
- Warn if incoming duplicate packets are more than 15 percent.
 - It is a problem if duplicate packets are at 25 percent; the remote system is retransmitting to you.
- Warn if listen drops are seen, it is a problem if drops are more than 0.5 per second.
- Warn if outgoing connection fails over 2 per second.

TCP Stack Issues

The overhead image above gives a number of tuning guidelines for the TCP/IP stack. “Warn” means that this is a condition that requires further investigation. The tuning parameters for these conditions will be discussed on the next pages.

There are many other tuning parameters for the TCP stack; do not adjust them. They are intended for development use or for conformance with rarely used parts of the TCP specification (RFC [request for comments] 793) and should not be used.

Make sure you have the latest patches for your TCP stack and network drivers. This area of the system changes constantly, and new features, especially performance features, are added regularly. Be sure to read the `README` files for details of the changes.



TCP Tuning Parameters

- `tcp_time_wait_interval` (was `tcp_close_wait_interval` prior to Solaris 2.6 OS)
 - Default 240,000 ms is reduced to 60,000 for SPECweb96 runs.
- `tcp_conn_req_max_q0` – Incomplete listen connection limit
 - Default is 1024; no need to tune unless `tcpListenDropQ0` is nonzero.

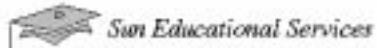
TCP Tuning Parameters

As mentioned before, there are many parameters that can be tuned in the TCP/IP stack. (For a complete list, run the `ndd /dev/tcp \?` command.) Similar parameters may be available for network adapters.

The close-time interval (closed socket reuse delay), with a default of 240 seconds (4 minutes) specifies how long a socket must remain idle until it can be reused. When changed to 1 minute, on systems such as web servers that use many sockets, significant response time improvements can be seen. Do not set it to less than 60,000 ms (1 minute.)

The request queue limit, the number of system-wide pending connection requests, is 1024 by default. Unless you are seeing dropped connection requests, which might occur on a heavily loaded HTTP server, do not adjust this parameter.

The pending listen queue can have problems similar to the incomplete



Using ndd

- Network driver parameters can be listed and changed with ndd.
- ndd can be run by anyone, but changes can be made only by the root user.
 - Changes last until reboot.
 - Permanent changes can be made through /etc/system.
- To see available parameters for a driver use:
`ndd /dev/tcp \?`
- Any network driver can be specified.

Using ndd

The `ndd` command enables you to inspect and change the settings on any driver involved in the TCP/IP stack. For example, you can look at parameters from `/dev/tcp`, `/dev/udp`, and `/dev/ip`. To see what parameters are available (some are read-only), use the `ndd /dev/ip \?` command. (The `?` is escaped to avoid shell substitution.)

To see the value of a parameter, specify the parameter name.

```
ndd /dev/ip ip_forwarding
```

Changes can be made by using `ndd -set`, such as:

```
ndd -set /dev/ip ip_forwarding 0
```

which disables IP forwarding until the system is rebooted. (This has the same effect as creating `/etc/notrouter`.) A persistent change can be made using `/etc/system`. The change made in `/etc/system` would look like:

```
set ip:ip_forwarding=0
```

ndd can be used by any user to inspect parameters, but only root can set them.

If you are changing parameters for a network interface, remember to specify the proper interface instance number. For example, to see the current parameter settings for interface hme3, use:

```
ndd /dev/hme instance 3 \?
```



Network Hardware Performance

- Networks are packet-switched buses.
- They have identical performance issues.
- Access to the wire is important, so low utilization is essential.
- Collisions imply there is not enough available access.
 - You can cut traffic or increase the network speed.
 - You may have a lot of unnecessary traffic.
 - NFS attribute messages may be unnecessary.

Network Hardware Performance

Remember, networks act like packet-switched buses, and have the same performance that other packet-switched buses, such as the SCSI bus, have.

The main issue with these buses is access to transmit requests and responses. While SCSI and some network topologies (such as Token Ring) have strict contention management protocols, others, such as Ethernet, operate on a first come, first served basis.

Ethernet, for example, begins transmitting when the network appears quiet. If another message is received during the transmission, the transmission is stopped and the other message continues through. This event, a collision, requires that the transmitter delay and try again. For each collision that occurs, the transmitter delays more.

To avoid collisions, it is important that the network be available when a node needs to transmit. This can be done by using a higher speed network, but another solution is to investigate the network traffic to see if unnecessary messages are being sent.

For example, NFS clients assume that the files they are referencing may change, and before using any data they have cached, request the current file attributes from the server to ensure that the file has not changed. If the file is not going to change often, the attribute timeout (`actimeo`) on the NFS mount command (or in `/etc/vfstab`) may be increased to avoid a large number of unnecessary attribute request messages.



NFS

- Tune the client and server first
 - Remove any non-NFS-specific performance issues
 - Increase the size of the DNLC and inode cache
 - Solve server I/O and memory performance issues
 - Resolve network performance issues
 - Make sure `actimeo` is not too low
 - Use `cachefs`
 - Use NFS Version 3; some protocol efficiencies were added

NFS

Before tuning NFS specifically, be sure that the client and server are properly tuned. Issues such as DNLC and inode caches that are too small can result in increased network traffic as the information has to be reread. Once the client and server are tuned, it becomes easier to identify specific problems as being NFS or network related.

The first step in network tuning is to try to eliminate as much message traffic as possible. Messages that are not sent provide the greatest performance and capacity improvement.

This can be done by using `cachefs` and the `actimeo` parameter, as well as properly sized client and server I/O caches.

Also, NFS Version 3 has some network message efficiencies and uses larger block sizes, which can add significant improvements to a very busy NFS environment. The `nfsstat -m` command from the client can identify which version of NFS you are using.



NFS Retransmissions

- If NFS server cannot respond to client request, the client retransmits.
- Retransmission rate is the major factor affecting NFS performance.
- `nfsstat -c` displays client retransmissions (*retrans*).
- If you have a high retransmission rate, check for:
 - Overloaded servers
 - Ethernet interface dropping packets
 - Network congestion

NFS Retransmissions

The NFS distributed file service uses an RPC facility which translates local commands into requests for the remote host. The RPCs are synchronous. That is, the client application is blocked or suspended until the server has completed the call and returned the results. One of the major factors affecting NFS performance is the retransmission rate.

If the file server cannot respond to a client's request, the client retransmits the request a specified number of times before it quits. Each retransmission imposes system overhead, and increases network traffic. Excessive retransmissions can cause network performance problems. If the retransmission rate is high, look for:

- Overloaded servers that take too long to complete requests
- An Ethernet interface dropping packets
- Network congestion which slows the packet transmission

Use the `nfsstat -c` command to show client statistics. The number of client retransmissions is reported by the *retrans* field.

```
# nfsstat -c

Client rpc:
Connection oriented:
calls      badcalls    badxids    timeouts   newcreds   badverfs
156274      4          0          0          0          0
timers      cantconn   nomem     interrupts
0          4          0          0
Connectionless:
calls      badcalls    retrans    badxids    timeouts   newcreds
9          1          0          0          0          0
badverfs   timers     nomem     cantsend
0          4          0          0

Client nfs:
calls      badcalls    clgets     cltoomany
153655      1          153655    0
Version 2: (6 calls)
null       getattr    setattr   root      lookup    readlink
0 0%       5 83%      0 0%     0 0%     0 0%     0 0%
read       wrocache   write     create    remove    rename
0 0%       0 0%      0 0%     0 0%     0 0%     0 0%
link       symlink    mkdir     rmdir    readdir  statfs
0 0%       0 0%      0 0%     0 0%     0 0%     1 16%
Version 3: (151423 calls)
null       getattr    setattr   lookup   access    readlink
0 0%       44259 29%  5554 3%  26372 17%  18802 12%  765 0%
read       write     create    mkdir    symlink  mknod
20639 13%  24486 16%  3120 2%  190 0%   4 0%    0 0%
remove    rmdir     rename   link    readdir  readdirplus
1946 1%   5 0%      1019 0%  17 0%   998 0%   764 0%
fsstat    fsinfo    pathconf commit
73 0%     33 0%     280 0%  2097 1%
#
```



NFS Server Daemon Threads

- There are a number of system daemons on NFS servers.
 - These handle client requests like mounts.
- I/O requests are handled by server system threads.
 - The default number of threads is 16.
- This is specified in `/etc/init.d/nfs.server`.
 - It must be adjusted based on server configuration.
 - You may need several hundred threads.
- A common cause of slow NFS servers is too few I/O daemon threads.

NFS Server Daemon Threads

NFS servers have several system daemons that perform operations requested by clients, such as `mountd` for mounting a shared file system to the client.

Normal client I/O requests, however, are handled by NFS I/O system threads. The NFS daemon, `nfsd`, services requests from the network. Multiple `nfsd` daemons are started so that a number of outstanding requests can be processed in parallel. Each `nfsd` takes one request off the network and passes it to the I/O subsystem.

The NFS I/O system threads are created by the NFS server startup script, `/etc/init.d/nfs.server`. The default number of I/O system threads is 16. To increase the number of threads, change the `nfsd` line found about halfway into the file.

```
/usr/lib/nfs/nfsd -a 16
```

One of the most common causes of poor NFS server performance is too few I/O daemon server threads. A large NFS server can require hundreds. You can intentionally supply too few I/O system threads to limit the server's NFS workload. If you configure too many `nfsds`, some may not be used, but it is unlikely that there will be any adverse side effects as long as you do not run out of process table entries (`max_nprocs`).

To determine the proper number of server threads, use the largest number calculated from the following:

- Two NFS threads per active client process
- Sixty-four NFS threads per SuperSPARC™ processor
- Two hundred NFS threads per UltraSPARC processor
- Sixteen threads per Ethernet interface
- One hundred and sixty threads per Fast Ethernet interface

Each `nfsd` system thread takes approximately 8 Kbytes of kernel memory.



Monitoring Network Performance

- Commands available for monitoring network performance include:
 - ping – Verifies connectivity to hosts on network
 - spray – Tests the reliability of packet sizes
 - snoop – Captures packets and traces calls
 - netstat – Displays network status
 - nfsstat – Displays NFS server and client statistics

Monitoring Network Performance

Commands available for monitoring network performance include:

- ping – Looks at the response of hosts on the network.
- spray – Tests the reliability of your packet sizes. It can tell you whether packets are being delayed or dropped.
- snoop – Captures packets from the network and trace the calls from each client to each server.
- netstat – Displays network status, including state of the interfaces used for TCP/IP traffic, the IP routing table, and the per-protocol statistics for UDP, TCP, ICMP (Internet control message protocol), and IGMP.
- nfsstat – Displays a summary of server and client statistics that can be used to identify NFS problems.

ping

Check the response of hosts on the network with the **ping** command. If you suspect a physical problem, you can use **ping** to find the response time of several hosts on the network. If the response from one host is not what you would expect, investigate that host. Physical problems can be caused by:

- Loose cables or connectors
- Improper grounding
- Missing termination
- Signal reflection

The simplest version of **ping** sends a single packet to a host on the network. If it receives the correct response, it prints the message host is alive.

```
# ping proto198
proto198 is alive
#
```

With the **-s** option, **ping** sends one datagram per second to a host. It then prints each response and the time it took for the round trip.

```
# ping -s proto198
PING proto198: 56 data bytes
64 bytes from proto198 (72.120.4.98): icmp_seq=0. time=0. ms
64 bytes from proto198 (72.120.4.98): icmp_seq=1. time=0. ms
64 bytes from proto198 (72.120.4.98): icmp_seq=2. time=0. ms
64 bytes from proto198 (72.120.4.98): icmp_seq=3. time=0. ms
^C
----proto198 PING Statistics----
5 packets transmitted, 4 packets received, 20% packet loss
round-trip (ms) min/avg/max = 0/0/0
#
```

spray

Test the reliability of your packet sizes with the `spray` command.

Syntax of the command is: # `spray [-c count -d interval -l packet_size] hostname`. Options are:

- `-c count` – Number of packets to send.
- `-d interval` – Number of microseconds to pause between sending packets. If you do not use a delay, you may run out of buffers.
- `-l packet_size` – The packet size.
- `hostname` – The system to send packets to.

The following example sends 100 packets to a host (`-c 100`); each packet contains 2048 bytes (`-l 2048`). The packets are sent with a delay time of 20 microseconds between each burst (`-d 20`).

```
# spray -c 100 -d 20 -l 2048 proto198
sending 100 packets of length 2048 to proto198 ...
no packets dropped by proto198
4433 packets/sec, 9078819 bytes/sec
#
```

snoop

To capture packets from the network and trace the calls from each client to each server, use snoop. snoop provides accurate time stamps that allow some network performance problems to be isolated quickly. Dropped packets could be caused by insufficient buffer space, or an overloaded CPU.

Captured packets can be displayed as they are received, or saved to a file for later inspection. Output from snoop can quickly overwhelm available disk space with the sheer amount of data captured. Tips to avoid information overload include:

- Capture output to a binary file

```
# snoop -o packet.file
```

```
^C#
```

- Display the contents of the binary file

```
# snoop -i packet.file
```

- Capture only relevant information such as the packet header contained in the first 120 bytes

```
# snoop -s 120
```

- Capture a specific number of packets

```
# snoop -c 1000
```

- Capture output to tmpfs to avoid dropping packet information due to disk bottlenecks

```
# snoop -o /tmp/packet.file
```

- Use filters to capture specific types of network activity

```
# snoop broadcast
```



Isolating Problems

- Check the four sources of problems:
 - Client
 - Server
 - Application
 - Network
- Ensure that the individual pieces are tuned, then look at the network and application specifically.

Isolating Problems

There are four sources of network problems, as shown in the overhead image above. By properly tuning the client and server systems, network and NFS performance can be significantly improved even before tuning the network itself.

Once the servers are operating correctly, and NFS activity has been tuned (for example, with `cachefs` and the `actimeout` parameter), look at physical network issues.

Just as system memory problems can cause I/O and CPU problems, system problems can cause network problems. By fixing any system problems first, you can get a much clearer look at the cause of the network problems, if they still exist.

Remember, you can use `ndd` and `ping`, `spray`, and `snoop` as well as `netstat` and `nfsstat` to get detailed information about your network and the network traffic.



Tuning Reports

- netstat:
 - Provides detailed physical traffic and error information
 - Shows detailed IP protocol-level message traffic
- nfsstat:
 - Provides detailed data transfer information
 - Gives detailed information for all NFS messages
 - Report meanings can change based on the network topology and protocols used.

Tuning Reports

The `netstat` monitor provides information on the physical network's performance as well as message and error counts from the TCP/IP stack.

The `nfsstat` monitor provides statistics on NFS data movement, file system usage, and NFS message types being transmitted. As of the Solaris 2.6 OS, `iostat` will report I/O performance data for NFS mounted file systems.

Samples of the `netstat` and `nfsstat` reports are in Appendix F.

Network Monitoring Using SE Toolkit Programs

net.se

The **net.se** program output is like that of **netstat -i** plus collision percentage.

```
# /opt/RICHPse/bin/se /opt/RICHPse/examples/net.se
Name          Ipkts      Ierrs      Opkts      Oerrs      Colls  Coll-Rate
hme0         1146294           0     1077416           0           0      0.00 %
#
```

netstatx.se

The **netstatx.se** program is like **iostat -x** in format. It shows per-second rates like **netstat**.

```
# /opt/RICHPse/bin/se /opt/RICHPse/examples/netstatx.se
Current tcp minimum retransmit timeout is 200  Thu Jul 29 11:15:29 1999

Name   Ipkt/s  Ierr/s  Opkt/s  Oerr/s  Coll/s  Coll%  tcpIn    tcpOut  DupAck %Retran
hme0    162.0    0.0    151.6    0.0     0.0    0.00  145695  128576      0      0.00
^C#
```

nx.se

The **nx.se** program prints out **netstat** and **tcp** class information. NoCP means nocanput, which is a packet that is discarded because it lacks IP-level buffering on input. It can cause a TCP connection to time out and retransmit the packet from the other end. Defr is defers, an Ethernet metric that counts the rate at which output packets are delayed before transmission.

```
# /opt/RICHPse/bin/se /opt/RICHPse/examples/nx.se
Current tcp RtoMin is 200, interval 5, start Thu Jul 29 13:18:20 1999

13:18:40  Iseg/s  Oseg/s  InKB/s  OuKB/s  Rst/s  Atf/s  Ret%  Icn/s  Ocn/s
tcp        1.6     1.6     0.19    0.18    0.00    0.00    0.0     0.00    0.00
Name      Ipkt/s  Opkt/s  InKB/s  OuKB/s  IErr/s  OErr/s  Coll%  NoCP/s  Defr/s
hme0      2.4      2.2     0.33    0.30    0.000   0.000   0.0     0.00    0.00
^C#
```

netmonitor.se

The netmonitor.se program waits for a slow network, then prints out the same basic data as netstatx.se. Data displayed is in per-second rates.

nfsmonitor.se

The nfsmonitor.se program is the same as nfsstat-m.se except that it is built as a monitor. It prints out only slow NFS client mount points, ignoring the ones that are working fine. Only as root can run it. For the Solaris 2.5 OS, nfsmonitor.se reliably reports data only for NFS V2 (Version 2). For NFS V3 (Version 3), it always reports zeroes (as does the regular nfsstat -m command).

tcp_monitor.se

The tcp_monitor.se program is a GUI that is used to monitor TCP and has a set of sliders that enable you to tune some of the parameters when run as root.



Check Your Progress

Before continuing on to the next module, check that you are able to accomplish or answer the following:

- Describe the different types of network hardware
- List the performance characteristics of networks
- Describe the use of IP trunking
- Discuss the NFS performance issues
- Discuss how to isolate network performance problems
- Determine what can be tuned

Think Beyond

Consider the following:

- What other similarities are there between system buses, the SCSI bus, and networks?
- Why does the TCP/IP stack have so many tuning parameters?
- Why is tuning the NFS client and server more likely to improve performance than tuning the network?
- How can you be sure that you have a network problem and not a system issue?

Objectives

Upon completion of this module, you should be able to:

- List four major performance bottlenecks
- Describe the characteristics and solutions for each
- Describe several different ways to detect a performance problem
- Identify the type of performance problem that you have, and what tools are available to resolve it

Relevance



Discussion – The following questions are relevant to understanding the content of this module:

- What general assessments can you quickly make of a poorly tuned system?
- Is there an order to tuning solutions?
- Are there any common problems to look out for?

Additional Resources



Additional resources – The following references can provide additional details on the topics discussed in this module:

- Cockcroft, Adrian, and Richard Pettit. 1998. *Sun Performance and Tuning*. Palo Alto: Sun Microsystems Press/Prentice Hall.
- Wong, Brian L. 1997. *Configuration and Capacity Planning for Solaris Servers*. Palo Alto: Sun Microsystems Press/Prentice Hall.

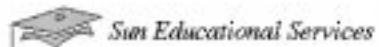


Some General Guidelines

- Tune one thing at a time
- Spend the most time tuning the things which have the largest effect; expect trade-offs
- Realize one size does not fit all
 - What works on a small system might not work on a larger system.
 - Many parameters have to be scaled or adjusted.
- Do not spend money first

Some General Guidelines

The overhead image above gives some basic guidelines for tuning a system.



What You Can Do

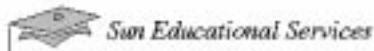
- Use accounting to see what is being run
 - Start accounting during boot
 - Use the `adm crontab` to process it
- Collect performance data regularly with the `sys crontab` entry
- Generate special purpose measurements with `sar`, `vmstat`, and so on
- Graph the data to see exceptions and trends
 - Consider using three-dimensional graphs

What You Can Do

You can

- Use accounting to monitor the workload. Enable accounting by using the following commands:

```
# ln /etc/init.d/acct /etc/rc0.d/K22acct
# ln /etc/init.d/acct /etc/rc2.d/S22acct
# /etc/init.d/acct start
# crontab -l adm
```
- Collect long-term system utilization data using `sys crontab`.
`# crontab -l sys`
- Graph your data. It is much easier to see trends and utilization problems in a graph than in columns of numbers. A good package to build on, `mrtg`, can be obtained from
<http://ee-staff.ethz.ch/~oetiker/webtools/mrtg/mrtg.html>. It can be easily modified to report on more than network data.



Application Source Code Tuning

It is usually much more productive to tune applications than to change OS parameters. (But do both.)

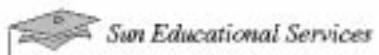
Application performance is affected by :

- Programming model
- Algorithms
- Choice of programming language
- Effective use of provided system functions
- Compiler and optimizations

Application Source Code Tuning

It is usually much more productive to tune applications than to change OS parameters. (But you can do both.)

Application performance is affected by many things. Review these, if possible, if you are experiencing application performance problems. You might not be able to fix them, but at least you will be able to identify the source of the problem, perhaps for future repair.



Compiler Optimization

- Can provide performance improvements of several hundred percent
- Is usually not done by the programmer
 - Makes programs a bit harder to debug
- Provides even better optimization for specific CPUs
- Use the basic compile flags -xO3 -xdepend
 - Compiles will run longer
- Use optimized libraries if available

Compiler Optimization

Remember, the majority of performance benefit comes from tuning the application. One of the simplest and most overlooked application performance improvements is the use of compiler optimization.

Most applications are compiled extremely inefficiently, for debugging purposes, and put into production that way. If compiled with basic optimization, performance gains of several hundred percent or more are possible, depending on the application.

Also, if you use analytical libraries, or shared libraries at all, be sure to compile them with optimization since their usage affects many other programs.

The slight increased difficulty in determining the location of a failure should be outweighed by significantly better performance.



Application Executable Tuning Tips

- Use accounting to see what is using resources
- Trace system calls (`truss`) to see what is going on
- Use `cachefs` to increase the speed of NFS reads
- Set the proper values for your basic tuning parameters

Application Executable Tuning Tips

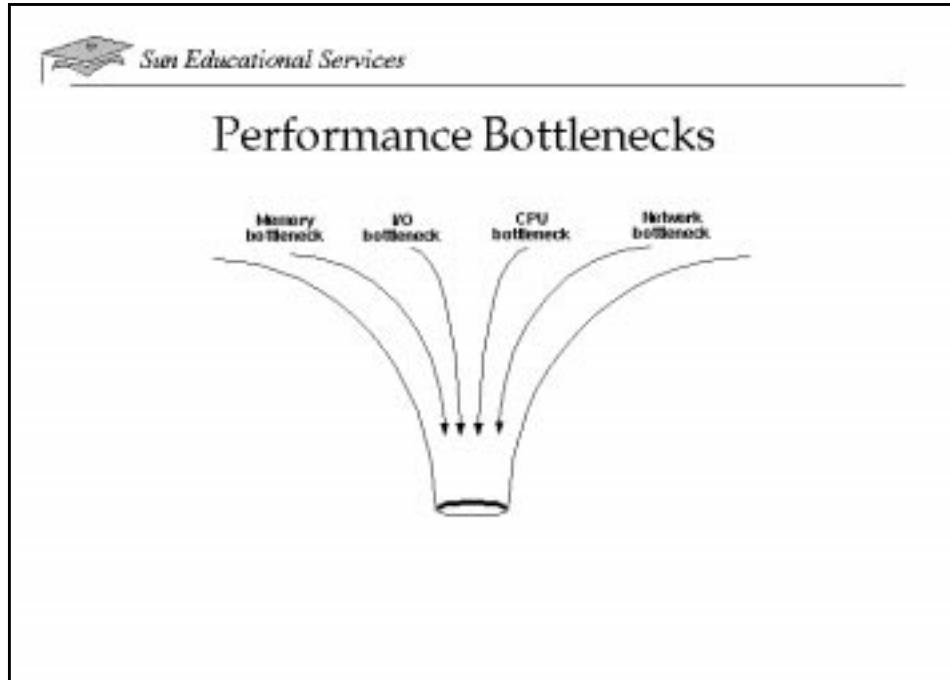
The overhead image gives a number of suggestions for locating problems with applications.

There might be difficulty initially determining which application is causing the problem, which system accounting can help with, or what the application is doing, which `truss` can help locate.

There may be NFS problems, incorrect system tuning parameters, and inefficient resource usage causing problems.

Also, application, DBMS, and system maintenance can introduce new functions or parameters, which might cause performance changes.

Indeed, almost anything could be wrong, which makes it important to be able to identify possible causes of the problem.



Performance Bottlenecks

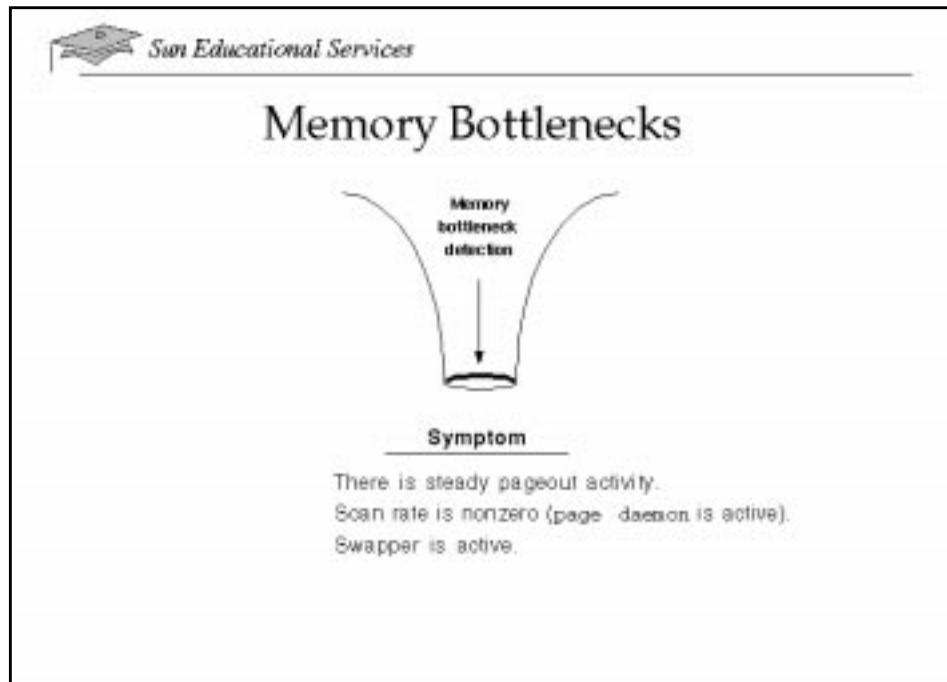
This section reviews the four basic performance bottlenecks: memory, I/O, CPU, and the network. This module summarizes the detection and removal of these bottlenecks.

The main monitors used to record performance data are sar, iostat, and vmstat. These performance monitors collect and display performance data at regular intervals. Selecting the sample interval and duration to use is partially dependent on the area of performance that is being analyzed.

Remember, long sample durations are useful for determining general trends over a large period of time. For example, to determine disk load, it may be more effective to sample over a longer period of time using larger intervals between samplings. If too short of a sample period is used, the data may not be representative of the general case (that is, the sample period might have occurred when the machine was abnormally busy).

Short sample durations are useful when the effects on the system of a specific application, device, and so on, are being determined. A shorter interval should be used so that more detailed data is recorded.

Remember, the data reported by these performance monitors is an average of the activity over the interval. The longer the interval, the less chance that a brief spike in performance will be noticed. However, the shorter the interval, the more often the performance monitor will run and, therefore, the more it skews the performance data being displayed.



Memory Bottlenecks

If memory is a bottleneck on a system, this means that there is not enough memory for all of the active threads to keep their working sets resident. When the amount of free memory drops below *lotsfree*, the page daemon will begin trying to free up pages that are not being used. If the amount of free memory drops too low, the swapper will begin swapping LWPs and their associated processes.

When the page daemon and the swapper need to execute, this will decrease the amount of execution time the application receives. If the page daemon is placing pages that have been modified on the page-free list, these pages will be written out to disk. This will cause a heavier load on the I/O system, up to the *maxpgio* limit.

The only utility that provides statistics on the hardware cache is `vmstat -c`. Cache misses are a potential bottleneck if they occur frequently enough. However, even when using `vmstat -c`, they are extremely difficult to detect.



Memory Bottleneck Solutions

- Adjust paging parameters such as `lotsfree`
- Use priority paging
- Use shared libraries
- Use `madvise` to use memory more efficiently
- Analyze locality of reference in applications
- Set memory usage limits with `ulimit`, `limits`, and `setrlimit(2)`
- Modify the process load if possible
- Add more physical memory

Memory Bottleneck Solutions

There are many possible solutions to a memory bottleneck, depending on the cause. Your main goal is to make more efficient use of existing memory. If a system is consistently low on memory, there are a few potential remedies that can be applied.

If several large processes are in execution at the same time, these processes might be contending with each other for free memory. Running these at separate times may lower the demand. For example, running several applications sequentially might put less of a load on the system than running all of them concurrently.

`memcntl` (or related library calls such as `madvise`) can be used to more effectively manage memory being used by an application. These routines can be used to notify the kernel that the application will be referencing a set of addresses sequentially or randomly. They can also be used to release memory that is no longer needed by the application back to the system.

lotsfree, *minfree*, *slowscan*, and *fastscan* can be used to control when the kernel begins paging and swapping and at what rate. They can be used to modify the impact the page daemon and swapper have on system performance.

Applications that use shared libraries require less physical memory than applications that have been statically linked if sharing takes place.

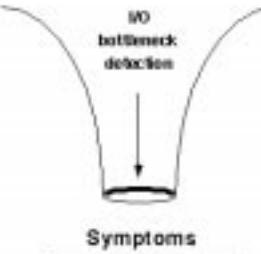
Functions that call one another within an application should be placed as close together as possible. If these functions are located on the same page, then only one physical page of memory is needed to access both functions.

`setrlimit` can be used to force applications to keep their stack and data segments within a specific size limit.

Adding more physical memory, or at least more and faster swap drives, is always a solution as well.

 Sun Educational Services

I/O Bottlenecks



Symptoms

- There is an uneven workload.
- Many threads are blocked waiting on I/O.
- The disk utilization rate is high.
- You have active disks with no free space.

I/O Bottlenecks

`sar`, `vmstat`, and `iostat` can be used to detect a possible I/O bottleneck.

One of the goals in this area is to spread the load out evenly amongst the disks on a system. This might not always be possible. But in general, if one disk is much more active than other disks on the system, there is a potential performance problem. Although determining an uneven disk load might vary, depending on the system and expectations, 20 percent is normally used. If the number of operations per disk vary by more than 20 percent, the system might benefit by having the disk load balanced.

If several threads are blocked waiting for I/O to complete, the kernel might not be able to service the requests fast enough to keep them from piling up. The time a thread spends waiting for an I/O request to complete is time that the thread could be in execution if the requested data was available. There is no field that records this information in `sar`. However, the `b` column in `vmstat` displays this information.

Disks that are busy a large percentage of time are also indications that there is a potential performance problem. There are two ways to measure this. If the number of reads per second plus the number of writes per second is greater than 65 percent of the disk's capacity, the disk might be overloaded. If the disk is busy more than 85 percent of the time, this may also indicate a disk load problem.

Because block allocation dramatically slows when a disk is nearly full, active disks with very little free space (near or less than the partition *minfree* value) might be slowing I/O due to space allocation overhead.



I/O Bottleneck Solutions

- Adjust the DNLC and inode cache sizes
- Set appropriate UFS partition parameters
- Balance your disk load
 - Use RAID-0 (striping) and proper disk/bus balance
 - Use mmap or direct I/O instead of plain read and write
 - Use shared libraries and compiler optimization
 - Organize I/O requests to be more contiguous
 - Move slow devices off of production I/O buses
 - Add more or faster disks or buses

I/O Bottleneck Solutions

Several actions can be taken to avoid I/O creating a performance bottleneck on a system.

A balanced disk load is very effective. The goal is to create a configuration where all disks on average are equally busy. If a system has some partitions which are heavily accessed, these partitions can be placed on separate disks. The number of disks per controller should also be considered when evaluating disk load. Several extremely fast disks all attached to a slow controller will not be very effective.

Remember, placing too many drives on a bus will overload that bus, causing poor response times that are not caused by the drives themselves.

Spreading swap partitions/files out among the disks will also be effective in balancing the disk load. When allocating swap space, the kernel works in round-robin fashion, allocating 1 Mbyte worth of space from a swap device/file before moving to the next swap device.

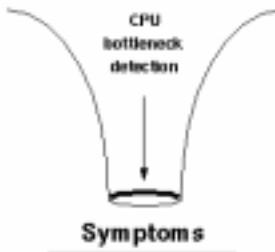
A large disk will, by virtue of its size, have more requests to service than a smaller disk. By placing busy file systems on smaller disks, the load between large and small disks can be more easily balanced.

`mmap` requires less overhead to do I/O than `read` and `write`, as does direct I/O.

Applications can be modified to be more efficient with respect to their I/O requests. Instead of making many smaller non-contiguous requests, an application should attempt to make fewer, larger requests. This is because in most cases, the overhead to actually find the data on disk is much larger than the overhead required to transfer the data, as you have seen.



CPU Bottlenecks



- CPU idle time is low.
- Threads are waiting on run queue.
- There is lower response time or interactive performance.

CPU Bottlenecks

The determination of whether the CPU is a performance bottleneck is largely a subjective one. This determination is based on the desired performance from the system. The user who wants rapid response time will use a different criteria than the user who is more interested in supporting a large number of jobs that are not interactive.

If the CPU is not spending a lot of time idle (less than 15 percent), then threads which are runnable have a higher likelihood of being forced to wait before being put into execution. In general, if the CPU is spending more than 70 percent in user mode, the application load might need some balancing. In most cases, 30 percent is a good high-water mark for the amount of time that should be spent in system mode.

Runnable threads waiting for a CPU are placed on the dispatch queue (or run queue). If threads are consistently being forced to wait on the dispatch queue before being put into execution, then the CPU is impeding performance to some degree. For a uniprocessor, more than two threads waiting on the dispatch queue might signal a CPU bottleneck. A run queue more than four times the number of CPUs indicates processes are waiting too long for a slice of CPU time. If this is the case, adding more CPU power to the system would be beneficial.

A slower response time from applications might also indicate that the CPU is a bottleneck. These applications may be waiting for access to the CPU for longer periods of time than normal. Consider locking and cache effects if no other cause for the performance problem can be found.



CPU Bottleneck Solutions

- Modify applications to use system calls more efficiently
 - Use `mmap` instead of `read` and `write`
- Use compiler optimization
- Use processor sets
- Use `priocntl` and `nice` to modify process priorities
- Modify the dispatch parameter tables
- Check old custom device drivers for PIO usage
- Modify or limit the process load
- Add more or faster CPUs

CPU Bottleneck Solutions

`priocntl` and `nice` can be used to affect a thread's priority. Since threads are selected for execution based on priority, threads that need faster access to the CPU can have their priority raised. You can also use processor sets to guarantee or limit access to CPUs.

The dispatch parameter tables can be modified to favor threads that are CPU intensive or I/O bound. They could also be modified to favor threads whose priorities fall within a specific range of priorities.

The CPU bottleneck might be the result of some other problem on the system. For example, a memory shortfall will cause the page daemon to execute more often. Other lower priority threads will not be able to run while the page daemon is in execution. Checking for system daemons which are using up an abnormally large amount of CPU time may indicate where the real problem lies.

`vmstat -i` should be used to determine if a device is interrupting an abnormally large number of times. Each time a device interrupts, the kernel must run the interrupt handler for that device. Interrupt threads run at an extremely high priority.

Lowering the process load by forcing some applications to run at different times might help. Modifying `max_nprocs` will reduce the total number of processes that are allowed to exist on the system at one time.

Any custom device drivers or third-party software should be checked for possible inefficiencies, especially unnecessary programmed I/O (PIO) usage with faster CPUs.

Lastly, if all else fails, adding more CPUs might alleviate some of the CPU load.



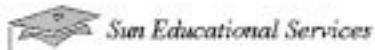
Ten Tuning Tips

1. Remember the interrelationships in the system. The source of a problem is not always obvious.
2. *Sustained* high scan rates can indicate a memory shortage.
3. Ignore free memory numbers; they should be about *lotsfree*.
4. Ignore paging I/O statistics.
5. Always check for disk bottlenecks.

Ten Tuning Tips

Some of the most useful tuning reminders are given on these next two pages.

Remember, do not overreact to a high value in a tuning report. There will always be transient periods when demands are higher on the system than usual, and usually it is not cost effective to handle them. Make sure, however, that you have enough system resources to support your regular workload with room for growth.



Ten Tuning Tips

6. Use `nfsstat -c` on slow NFS clients to find the source of the problem.
7. Use priority paging.
8. Graph your data to see exceptions quickly.
9. Base your tuning on the characteristics of the component (bus or cache).
10. Things change: check patches and new release documentation.

Do not hesitate to call your application vendor if you suspect problems with an application system. They are the experts on their system and might be able to quickly suggest a solution that might take days to find any other way.

Case Study

The following case study involves a customer who is experiencing less than satisfactory system performance with a statistical analysis software package (SAS) running on Sun equipment. The customer asked Sun personnel to evaluate the system and recommend solutions to the performance problems.

When attempting to performance tune a system, it is necessary to gather information regarding the system configuration, the applications that are running on the system, and information on the user community, such as the number of users, where the users are located, and what type of work patterns they have.

Some initial questions need to be answered. Data needs to be collected to confirm a performance problem and then, solutions must be implemented. It is important to make one change at a time and then reevaluate performance before making any other changes.

This particular customer has three Sun Enterprise 6000 servers. Two of the servers are running Oracle®, a relational database management system, and the third server is running an SAS application. Each of the three servers is connected to two different storage arrays. Each of the disk arrays has 1.25 Tbytes of mirrored storage capacity.

Presently, 200 plus users have been created on the SAS server. The users have remote access to the server over 100BaseT Ethernet. Each user is connected to the server through a Fast Ethernet switch which means each user has a dedicated 100-Mbyte/sec pipe into the local area network.

Each of the 200 users have cron jobs that query the Oracle database for data, perform some statistical analysis on the data and then, generate reports. The system is quiet at night and the cron jobs start at around 5:30 each morning and run throughout the day.

Initially, 50 users were set up and performance was fine at that point. System performance degraded as more users were added. Queries to the Oracle database were still satisfactory, but performing the statistical analysis on the SAS server was taking longer as more and more users were added to the system. It was the SAS server that was running slow.

Case Study – System Configuration

The **prtdiag** utility, first provided with the SunOS 5.5.1 operating system, provides a complete listing of a system's hardware configuration, up to the I/O interface cards.

Partial output of the **prtdiag** command follows. This particular system is a Sun Enterprise 6000 with eight CPUs and 6 Gbytes of memory.

```
# /usr/platform/sun4u/sbin/prtdiag
```

```
System Configuration: Sun Microsystems sun4u 16-slot Sun Enterprise 6000
```

```
System clock frequency: 84 MHz
```

```
Memory size: 6144Mb
```

```
===== CPUs =====
```

Brd	CPU	Module	Run MHz	Ecache MB	CPU Impl.	CPU Mask
0	0	0	336	4.0	US-II	2.0
0	1	1	336	4.0	US-II	2.0
2	4	0	336	4.0	US-II	2.0
2	5	1	336	4.0	US-II	2.0
4	8	0	336	4.0	US-II	2.0
4	9	1	336	4.0	US-II	2.0
6	12	0	336	4.0	US-II	2.0
6	13	1	336	4.0	US-II	2.0

```
===== Memory =====
```

Brd	Bank	MB	Status	Condition	Speed	Intrlv. Factor	Intrlv. With
0	0	1024	Active	OK	60ns	4-way	A
0	1	1024	Active	OK	60ns	2-way	B
2	0	1024	Active	OK	60ns	4-way	A
2	1	1024	Active	OK	60ns	2-way	B
4	0	1024	Active	OK	60ns	4-way	A
6	0	1024	Active	OK	60ns	4-way	A

...

...

#

The prtdiag I/O section shows, by name and model number, the interface cards installed in each peripheral I/O slot.

```
# /usr/platform/sun4u/sbin/prtdiag
...
...
=====
          IO Cards =====

      Bus   Freq
Brd Type MHz   Slot  Name
----- -----
 1 SBus 25    0    nf
 1 SBus 25    1    fca
 1 SBus 25    2    fca
 1 SBus 25    3    SUNW,hme
 1 SBus 25    3    SUNW,fas/sd (block)
 1 SBus 25   13   SUNW,socal/sf (scsi-3)      501-3060
 3 SBus 25    0    QLGC,isp/sd (block)        QLGC,ISP1000
 3 SBus 25    1    fca
 3 SBus 25    2    fca
 3 SBus 25    3    SUNW,hme
 3 SBus 25    3    SUNW,fas/sd (block)
 3 SBus 25   13   SUNW,socal/sf (scsi-3)      501-3060
 5 SBus 25    0    QLGC,isp/sd (block)        QLGC,ISP1000
 5 SBus 25    3    SUNW,hme
 5 SBus 25    3    SUNW,fas/sd (block)
 5 SBus 25   13   SUNW,socal/sf (scsi-3)      501-3060
 7 SBus 25    0    fca
 7 SBus 25    2    QLGC,isp/sd (block)        QLGC,ISP1000
 7 SBus 25    3    SUNW,hme
 7 SBus 25    3    SUNW,fas/sd (block)
 7 SBus 25   13   SUNW,socal/sf (scsi-3)      501-3060

No failures found in System
=====

No System Faults found
=====

#
```

The most serious bottlenecks are usually I/O-related. Either the system bus is overloaded, the peripheral bus is overloaded, and/or the disks themselves are overloaded.

From the information in the I/O section of the `prtdiag` command, the requested bandwidth for the bus can be calculated. This will allow you to determine if I/O bottlenecks are caused by installing interface cards with more bandwidth capability than the peripheral bus can handle. Given the maximum bus bandwidths for the system, and the bandwidths necessary for the interface cards, it should be easy to tell if the available bandwidth for the peripheral bus has been exceeded.

On the Enterprise 6000, each board actually has two SBuses for an aggregate bandwidth of 200 Mbytes/sec. The bandwidth of the Gigaplane bus on the E6000 is 2.6 Gbytes/sec. With the four boards (1, 3, 5, and 7) each having two SBuses, aggregate bandwidth of the eight SBuses is 800 Mbytes/sec. This is well below the available bandwidth of the 2.6-Gbyte backplane.

Since the SBuses on board 1 have the most devices, this board will be evaluated for an overload condition. Following are the bandwidths for the devices on board 1:

- nf – FDDI (fiber distributed data interface) at 100 Mbps (or 12.5 Mbytes/sec)
- fca – Fibre Channel controller at 40 Mbytes/sec
- fca – Fibre Channel controller at 40 Mbytes/sec
- SUNW,hme – Fast Ethernet at 100 Mbps (or 12.5 Mbytes/sec)
- SUNW,fas/sd – Fast/Wide SCSI at 20 Mbytes/sec
- SUNW,socal/sf – Serial Optical Channel at 100 Mbytes/sec

Totaling the bandwidths results in 225 Mbytes/sec which does exceed the maximum bandwidth of the SBuses. With a peak request for 225 Mbytes/second bandwidth and at most, 200 Mbytes/sec available, delays are inevitable. The customer was advised to move an fca to board 5. The I/O cards on board 5 as currently configured, only have an aggregate bandwidth of 152.50 Mbytes/sec. Moving one fca to board 5 will result in a bandwidth of 145 Mbytes/sec for board 1 and 192.5 Mbytes/sec on board 5.

Case Study – Process Evaluation

Once you identify which processes are active, you can narrow it down to the one(s) causing any delays and determine which system resources are causing bottlenecks in the process. The BSD version of the `ps` command displays the most active processes in order.

```
# /usr/ucb/ps -aux
USER      PID %CPU %MEM   SZ   RSS TT      S      START TIME COMMAND
user1     3639 11.1  0.120376 4280 pts/9      O 06:42:22 115:37 sas
user2     22396 8.2   0.1 6104 4208 ?          O 09:04:13 11:51 /usr/local/sas612/
user3     24306 5.0   0.53086428976 ?          S 09:14:57  6:10 /usr/local/sas612/
user4     21821 4.9   0.1 7656 4608 ?          S 09:00:07  4:56 /usr/local/sas612/
user5     17093 2.7   0.1 8328 5080 ?          S 08:25:55  0:46 /usr/local/sas612/
user6     20156 2.7   0.218896 7480 pts/14      O 08:47:05  4:26 sas
user7     12982 2.2   0.1 5576 3680 ?          O 08:00:01 11:02 /usr/local/sas612/
user8     25108 2.0   0.1 5896 4152 ?          S 09:18:45  0:26 /usr/local/sas612/
user9     24315 1.7   0.43369622920 ?         O 09:15:01  2:21 /usr/local/sas612/
#
#
```

You knew that SAS would be the most active process on the system, however, the report still provides some useful information. The `%CPU` column is the percentage of the CPU time used by the process. As you can see, the first SAS process is using a large percentage of CPU time, 11.1 percent.

Note – User IDs were changed in the `ps` output to protect the privacy of the actual users.

The status column, `S`, is also useful. A status of `P` indicates that the process is waiting for a pagein. A status of `D` indicates that the process is waiting for disk I/O. Either of these statuses could indicate an overload of the disk and memory subsystems.

This is not the case on this system. The processes have a status of `O` indicating it is on the CPU, or running. A status of `S` indicates that the process is sleeping.

The process identifier (PID) of the active processes is also useful. The memory requirements of the process will be determined by using the `/usr/proc/bin/pmap -x PID` command.

Case Study – Disk Bottlenecks

An I/O operation cannot begin until the request has been transferred to the device. The request cannot be transferred until the initiator can access the bus. As long as the bus is busy, the initiator must wait to start the request. During this period, the request is queued by the device driver. A read or write command is issued to the device driver and sits in the wait queue until the SCSI bus and disk are both ready.

Queued requests must also wait for any other requests ahead of them. A large number of queued requests suggests that the device or bus is overloaded. The queue can be reduced by several techniques: use of tagged queueing, a faster bus, or moving slow devices to a different bus.

Queue information is reported by the `iostat -x` report. The queue length, that is, the average number of transactions waiting for service, is reported by the `wait` column. The percentage of time the queue is not empty; that is, the percentage of time there are transactions waiting for service is reported by the `%w` column. If a system spends a good percentage of time waiting for I/O, this could indicate a disk bottleneck. To locate the device(s) causing a high wait I/O percentage, check the `%w` column.

```
# iostat -x
                                extended device statistics
device    r/s    w/s    kr/s    kw/s    wait   actv   svc_t    %w    %b
sd0      9.4    2.1    79.2    57.0    0.4    0.5    75.0    1     9
sd1      2.5    3.1    62.3   140.4    0.0    0.2    34.0    0     5
sd2      3.1    3.8   121.4   166.9    0.1    1.6   246.8    1    21
sd3      1.4    0.2    71.0     8.4    0.0    0.0    25.6    0     3
sd4      3.1    3.6   120.0   164.0    0.0    1.0   156.2    1    18
sd5      1.4    0.2    69.9     8.4    0.0    0.0    25.6    0     3
sd6      3.0    3.7   118.6   163.9    0.1    1.6   258.9    2    21
```

As you can see from this random sampling of I/O devices, none of the devices report a high number of transactions waiting for service. The percentage of time there are transactions waiting for service is also quite acceptable. This suggests that the devices are distributed properly and that the bus is not overloaded.

You can also use `iostat -x` to look for disks that are more than 5 percent busy, `%b`, and have average response times, `svc_t`, of more than 30 ms. Careful configuration, that is, spreading the load and using a disk array controller that includes NVRAM (non-volatile random access memory) can keep average response time under 10 ms.

```
# iostat -x
```

device	extended device statistics									
	r/s	w/s	kr/s	kw/s	wait	actv	svc_t	%w	%b	
sd0	9.4	2.1	79.2	57.0	0.4	0.5	75.0	1	9	
sd1	2.5	3.1	62.3	140.4	0.0	0.2	34.0	0	5	
sd2	3.1	3.8	121.4	166.9	0.1	1.6	246.8	1	21	
sd3	1.4	0.2	71.0	8.4	0.0	0.0	25.6	0	3	
sd4	3.1	3.6	120.0	164.0	0.0	1.0	156.2	1	18	
sd5	1.4	0.2	69.9	8.4	0.0	0.0	25.6	0	3	
sd6	3.0	3.7	118.6	163.9	0.1	1.6	258.9	2	21	

From the `iostat` report, you can see that average response times are high for all disks. Also, `sd0`, `sd2`, `sd4`, and `sd6` are busy more than 5 percent of the time. A better distribution of load across these disks would help. Reducing the number of I/Os would also help.

UFS tries to balance the performance gained by write-back caching with the need to prevent data loss. Main memory is treated as a write-back cache, and UFS treats the `segmap` cache the same way. The `fsflush` daemon periodically wakes up and writes modified file data to disk. Without this mechanism, modified file data could remain in memory indefinitely.

There are two tuning parameters that control the `fsflush` daemon:

- `tune_t_fsflushr` – How often to wake up the `fsflush` daemon. The default is 5 seconds.
- `autoup` – How long a cycle should take; that is, the longest that unmodified data will remain unwritten. Default is 30 seconds.

The SAS application reads data from the Oracle database tables, performs some statistical analysis, and produces reports. Therefore, it is not necessary to activate the `fsflush` daemon as often. SAS requires a working file system for each user to write the results of computations to disk. But these writes are not that critical because the original Oracle data can always be read again in the event of a system crash.

Increasing both tuning variables for the `fsflush` daemon will decrease the number of CPU cycles required to run `fsflush`, resulting in an increase in the number of CPU cycles for SAS. Increasing both tuning variables will also decrease the number of I/Os required to write file data back to disk. Recommended entries in `/etc/system` are:

```
set tune_t_fsflushr=10
set autooup=120
```

A blocked process is a sign of a disk bottleneck. The number of blocked processes should be less than the number of processes in the run queue. Whenever there are any blocked processes, all CPU idle time is treated as wait for I/O time. Use the `vmstat` command to display run queue (*r*) blocked processes (*b*) and CPU idle time (*id*).

```
# vmstat 2
procs      memory          page          disk          faults      cpu
r b w    swap  free   mf   pi   po   fr de sr s0 s1 s2 s3 in sy cs us sy id
1 1 34 121984 41312 13 2612 11218 2439 14303 0 1528 12 0 0 6 1425 2872 926 15 21 63
0 0 71 6866280 97728 37 4056 22808 11392 44380 0 4220 2 0 0 0 2391 5459 969 22 46 32
0 1 71 6866280 94832 49 5049 28232 12948 47268 0 4347 6 1 0 0 2345 5559 1076 24 53 22
0 2 71 6868128 92984 5 4935 31444 20432 55256 0 4467 13 0 0 0 2364 5530 1056 27 44 29
0 1 71 6868416 95480 5 4621 30732 20100 53460 0 4287 4 0 0 0 2637 5138 1175 24 44 32
^C#
```

In this case, the number of blocked processes exceeds the number in the run queue indicating that the disk subsystem requires additional tuning. Continue to balance devices across the bus, balance the load across disk devices, and decrease the number of I/Os. Increasing the inode cache size (`ufs_ninode`) may also help reduce the number of disk I/Os.

Case Study – CPU Utilization

In order to collect data on all processes, system accounting was enabled. Performance monitoring data could then be collected on a regular basis and analysis of system resource usage performed.

The `sar -u` command reports CPU utilization. Output from `sar -u` confirms the user's complaints that the system begins to be sluggish at around 8:00 or 9:00 in the morning. As you can see from the report, `%idle` (percentage of time the processor is idle) was zero. As has been discussed previously, zero idle time is a good sign that a system is overloaded. `%wio` (percentage of time the processor is idle and waiting for I/O completion) is also high after 9:00. Once again, this indicates that the I/O devices are a bottleneck.

```
# sar -u
```

```
SunOS pansco-sdm 5.6 Generic_105181-13 sun4u      07/15/99
```

	%usr	%sys	%wio	%idle
00:00:01				
01:00:01	5	8	62	26
02:00:01	0	3	13	84
03:00:01	1	3	1	95
04:00:00	4	3	1	91
05:00:01	3	4	2	91
06:00:01	10	13	5	72
07:00:03	37	53	4	6
08:00:05	33	61	5	0
08:20:02	33	63	5	0
08:40:01	34	66	0	0
09:00:02	32	66	2	0
09:20:02	33	54	12	1
09:40:01	32	48	20	0
10:00:01	35	48	17	0
10:20:01	26	32	39	3
Average	16	26	12	45
#				

Case Study – CPU Evaluation

On a multiprocessor system (which this system is), the `mpstat` utility is used to display per-processor performance statistics. This command can be used to detect load imbalances between CPUs.

On this system, the load distribution between CPUs is satisfactory. CPUs 4 and 5 appear to be working harder than the others. Adding more CPUs would not help improve performance, however, since each of the CPUs is spending at least some of the time idle (*idl*) already. The CPUs are spending a significant amount of time waiting for I/O (*wt*) which again indicates an I/O bottleneck. The number of interrupts (*intr*) are high which could also be attributed to I/O device interrupts.

Using processor sets, changing process priorities by modifying dispatch tables, running applications in realtime, or using the Solaris Resource Manager are all options to improve the efficiency of the CPUs. But these all require some trial-and-error. The customer, at this time, has elected not to experiment with any of these options.

```
# mpstat 2
CPU minf mjf xcal intr ithr csw icsw migr smtx srw syscl usr sys wt idl
 0 411 1 17351   61  40 106  20  12  85  0  418 19 21 21 39
 1 467 1 757    21  0 110  20  11  89  0  469 22 22 19 36
 4 431 1 17997   20  0 105  19  11  87  0  431 20 21 21 38
 5 467 1 59    206 183  92  18  10  91  0  452 23 22 19 37
 8 218 1 14856  245 207 126  19  14 106  0  273 10 20 26 44
 9 213 1 15165  249 210 130  19  14 112  0  282 10 20 26 44
12 189 1 14071  259 218 131  20  13 105  0  257  9 19 27 45
13 215 1 14944  461 226 123  19  13 107  0  288 10 24 23 42
CPU minf mjf xcal intr ithr csw icsw migr smtx srw syscl usr sys wt idl
 0 591 3 30545   14  0  94  12  18  82  0  480 27 27 28 18
 1 717 5 33174   11  1 108   9  18 120  0  399 12 32 38 19
 4 803 1 26498   13  0  50  12   8  85  0  687 67 24 10  0
 5 580 6 27443   96  80  89  15  18  66  0  397 46 24 28  3
 8 317 19 20689  426 386 199  14  26 145  1  411 10 20 54 16
 9  72 13 14290  411 368 140  14  24 121  0  249 20 14 44 22
12 135 2 12222  409 371 133   8  20 102  0  183   4  8 62 26
13 417 0 25322  580 349  93  10  17 124  0  166 14 36 34 16
^C#
```

Case Study – Memory Evaluation

The amount of total physical memory can be determined from the output of the `prtdiag` command (as shown previously) and also from the `prtconf` command. For this system, physical memory is 6 Gbytes.

```
# prtconf
System Configuration: Sun Microsystems sun4u
Memory size: 6144 Megabytes
System Peripherals (Software Nodes):
...
...
#
```

The amount of memory allocated to the kernel can be found by using the Solaris `sar -k` command and totaling all of the `alloc` columns. The output is in bytes.

```
# sar -k
```

```
SunOS pansco-sdm 5.6 Generic_105181-13 sun4u      07/15/99

00:00:01  sml_mem    alloc   fail   lg_mem    alloc   fail   ovsz_alloc   fail
01:00:01  68993024  51091032      0  228032512  150674684      0  29458432      0
02:00:01  69902336  44424744      0  228089856  114794740      0  29589504      0
03:00:01  70803456  37963860      0  228384768  79356972      0  29655040      0
04:00:00  64528384  32227740      0  139239424  71898524      0  29622272      0
05:00:01  64176128  31738052      0  138182656  72442176      0  29589504      0
06:00:01  63840256  32549792      0  134479872  74934108      0  30179328      0
07:00:03  62021632  33294448      0  118980608  76080604      0  30801920      0
08:00:05  61349888  33669428      0  117096448  75169864      0  31293440      0
08:20:02  61218816  34200836      0  116613120  75783280      0  31555584      0
08:40:01  61571072  38224384      0  115449856  97785736      0  31686656      0
09:00:02  61325312  38983276      0  114278400  98494340      0  31260672      0
09:20:02  60047360  36846884      0  114139136  86480376      0  31326208      0
09:40:01  58433536  34901304      0  113737728  76232028      0  32120832      0
10:00:01  58023936  34809740      0  113131520  74745224      0  31653888      0
10:20:01  57958400  33979004      0  112869376  74450408      0  31588352      0

Average   62946236  36593635      0  142180352  86621538      0  30758775      0
#
```

Totaling the averages of small, large, and oversize allocations, the kernel has grabbed 153 Mbytes of memory.

Knowing how much memory an application uses allows you to predict the memory requirements when running more users.

Use the `pmap` command or Memtool's `pmem` functionality to determine how much memory a process is using and how much of that memory is shared. The PID of the process, 3639, was obtained using the `ps` command.

Address	Kbytes	Resident	Shared	Private	Permissions	Mapped File
00002000	8	8	-	8	read	[anon]
00010000	2048	2000	1848	152	read/exec	sas
0021E000	56	32	8	24	read/write/exec	sas
0022C000	1296	16	-	16	read/write/exec	[heap]
ED870000	336	336	336	-	read/exec/shared	dev:158,34006 ino:3073
ED8C4000	8	8	-	8	read/write/exec	dev:158,34006 ino:3073
ED8C6000	16	8	-	8	read/write	dev:158,34006 ino:3073
ED8D0000	16	16	16	-	read/exec/shared	dev:158,34006 ino:3241
ED8D4000	8	-	-	-	read/write/exec	dev:158,34006 ino:3241
ED8E0000	296	24	8	16	read/exec/shared	dev:158,34006 ino:3169
ED92A000	8	-	-	-	read/write/exec	dev:158,34006 ino:3169
ED92C000	24	-	-	-	read/write	dev:158,34006 ino:3169
ED940000	256	32	-	32	read/write/exec	[anon]
ED990000	200	152	-	152	read/exec/shared	dev:158,34006 ino:3063
ED9C2000	8	-	-	-	read/write/exec	dev:158,34006 ino:3063
ED9C4000	24	24	-	24	read/write	dev:158,34006 ino:3063
ED9D0000	184	96	16	80	read/exec/shared	dev:158,34006 ino:3156
ED9FE000	8	-	-	-	read/write/exec	dev:158,34006 ino:3156
EDA00000	8	-	-	-	read/write	dev:158,34006 ino:3156
EDA10000	280	208	56	152	read/exec/shared	dev:158,34006 ino:3157
EDA56000	8	-	-	-	read/write/exec	dev:158,34006 ino:3157
EDA58000	96	80	-	80	read/write	dev:158,34006 ino:3157
EDA80000	512	16	-	16	read/write/exec	[anon]
EDB10000	8	8	8	-	read/exec/shared	dev:158,34006 ino:3116
EDB12000	8	-	-	-	read/write/exec	dev:158,34006 ino:3116
EDB20000	96	8	-	8	read/exec/shared	dev:158,34006 ino:3061
EDB38000	8	-	-	-	read/write/exec	dev:158,34006 ino:3061
EDB3A000	8	-	-	-	read/write	dev:158,34006 ino:3061
...						
...						
EF7D0000	112	112	112	-	read/exec	ld.so.1
EF7FA000	8	8	8	-	read/write/exec	ld.so.1
FFFF4000	48	16	-	16	read/write/exec	[stack]
-----	-----	-----	-----	-----	-----	-----
total Kb	20376	9904	6712	3192		

The output of the `pmap` command shows that the SAS process is using 9904 Kbytes of real memory. Of this, 6712 Kbytes is shared with other processes on the system, via shared libraries and executables.

The pmap command also shows that the SAS process is using 3192 Kbytes of private (non-shared) memory. Another instance of SAS will consume 3192 Kbytes of memory (assuming its private memory requirements are similar).

Memory requirements for the 200 users the customer has set up to run SAS would be 638,400 Kbytes (or 638 Mbytes). This system has 6 GBytes of memory which is more than enough to support 200 simultaneous SAS users.

The DNLC caches the most recently referenced directory entry names and the associated vnode for the directory entry. The number of name lookups per second is reported as *namei/s* by the sar -a command.

```
# sar -a
```

```
SunOS pansco-sdm 5.6 Generic_105181-13 sun4u      07/15/99

00:00:01  igure/s namei/s dirbk/s
01:00:01      1      240     1079
02:00:01      0      167     1108
03:00:01      0      167     1100
04:00:00      0      162     1079
05:00:01      0      176     1080
06:00:01      1      204     1063
07:00:03      1      231      906
08:00:05      3      238      836
08:20:02      1      223      788
08:40:01     58      296      609
09:00:02      4      281      734
09:20:02      2      245      910
09:40:01      1      208      844
10:00:01      0      227      842
10:20:01      0      233      962

Average       3      209      982
#
```

If namei does not find a directory name in the DNLC, it calls igure to get the inode for either a file or directory. Most igure calls are the result of DNLC misses. *igure/s* reports the number of requests made for inodes that were not in the DNLC.

vmstat -s will display the DNLC hit rate and lookups that failed because the name was too long. The hit rate should be 90 percent or better. As you can see, on this system, the hit rate is only 78 percent.

```
# vmstat -s

1156 swap ins
    916 swap outs
    2312 pages swapped in
    16226 pages swapped out
591157504 total address trans. faults taken
    53993307 page ins
    8768923 page outs
317283000 pages paged in
    69002475 pages paged out
    3089577 total reclaims
    3077302 reclaims from free list
        0 micro (hat) faults
591157504 minor (as) faults
    2286240 major faults
    4181274 copy-on-write faults
    22339837 zero fill page faults
345756783 pages examined by the clock daemon
    452 revolutions of the clock hand
404559907 pages freed by the clock daemon
    300274 forks
    9104 vforks
    252881 execs
209518071 cpu context switches
345103090 device interrupts
623248924 traps
649934363 system calls
156855213 total name lookups (cache hits 78%)
    181605 toolong
27810894 user    cpu
38464345 system  cpu
73582949 idle    cpu
41121115 wait    cpu
#
```

For every entry in the DNLC there will be an entry in the inode cache. This allows the quick location of the inode entry, without having to read the disk.

If the cache is too small, it will grow to hold all of the open entries. Otherwise, closed (unused) entries will remain in the cache until they are replaced. This is beneficial, for it allows any pages from the file to remain available in memory. A file can be opened and read many times, with no disk I/O occurring.

inode cache size is set by the kernel parameter *ufs_ninode*. You can see inode cache statistics by using netstat -k to dump out the raw kernel statistics information. *maxsize* is the variable equal to *ufs_ninode*. A *maxsize_reached* value higher than *maxsize* indicates the number of active inodes has exceeded cache size.

```
# netstat -k
kstat_types:
raw 0 name=value 1 interrupt 2 i/o 3 event_timer 4

segmap:
fault 1646083218 faulta 0 getmap 1531102064 get_use 8868192 get_reclaim 534564036
get_reuse 988319526
...
...
inode_cache:
size 3607 maxsize 17498 hits 770740 misses 591785 kmem allocs 95672 kmem frees 92065
maxsize reached 21216 puts at frontlist 847788 puts at backlist 316742
queues to free 0 scans 8873146 thread idles 587283 lookup idles 0 vget idles 0
cache allocs 591785 cache frees 666538 pushes at close 0
...
...
#
```

Case Study - Memory Tuning

Any nonzero values for `%ufs_ipf`, as reported by `sar -g`, indicate that the inode cache is too small for the current workload. `%ufs_ipf` reports flushed inodes, that is, the percentage of (closed) UFS inodes flushed from the cache which had reusable file pages still in memory.

```
# sar -g

SunOS pansco-sdm 5.6 Generic_105181-13 sun4u      07/15/99

00:00:01 pgout/s ppgout/s pgfree/s pgscan/s %ufs_ipf
01:00:01      0.16      0.26   1087.63   1055.39      0.00
02:00:01      0.16      0.27     77.66    83.49      0.00
03:00:01      0.15      0.27     82.27    72.65      0.00
04:00:00      0.27      1.27    134.35   137.28      0.00
05:00:01      0.87      5.98   100.98    84.64      0.00
06:00:01      1.89     11.58   539.67   525.53      0.00
07:00:03     33.47    262.36   2816.19   2584.70      0.00
08:00:05    176.68   1383.28   6086.32   4931.94      0.25
08:20:02    205.35   1614.82   6049.20   4643.43      0.22
08:40:01    188.46   1497.26   5477.24   4238.56      0.17
09:00:02    192.56   1521.26   5507.22   4215.31      0.18
09:20:02    113.15    883.76   4027.47   3246.40      0.10
09:40:01    127.03   1004.41   4296.40   3380.89      0.10
10:00:01     68.85    545.83   3400.88   2861.87      0.00
10:20:01     15.34    120.68   2090.83   1996.26      0.00

Average      50.05    392.96   2052.27   1709.92      0.07
#
```

Increasing the size of the DNLC cache, `ncsize`, and the size of the inode cache, `ufs_ninode`, will improve the cache hit rate and help reduce the number of disk I/Os. The inode cache should be the same size as the DNLC. Recommended entries in `/etc/system` are:

```
set ufs_ninode=20000
```

```
set ncsize=20000
```

The UFS buffer cache (also known as metadata cache) holds inodes, cylinder groups and indirect blocks only (no file data blocks). The default buffer cache size allows up to 2 percent of main memory to be used for caching the file system metadata. The default is too high for systems with large amounts of memory.

Use the `sysdef` command to check the size of the cache, `bufhwm`.

```
# sysdef | grep bufhwm  
126189568 maximum memory allowed in buffer cache (bufhwm)  
#
```

Since only inodes and metadata is stored in the buffer cache, it does not need to be a very large buffer. In fact, you only need 300 bytes per inode, and about 1 Mbyte per 2 Gbytes of files that are expected to be accessed concurrently. The size can be adjusted with the `bufhwm` kernel parameter, specified in Kbytes, in the `/etc/system` file.

For example, if you have a database system with 100 files totaling 100 Gbytes of storage space and you estimate that only 50 Mbytes of those files will be accessed at the same time, then at most you would need 30 Kbytes ($100 \times 300 \text{ bytes} = 30 \text{ Kbytes}$) for the inodes, and about 25 Mbytes ($50 \div (2 \times 1 \text{ Mbyte}) = 25 \text{ Mbytes}$) for the metadata. Making the following entry into `/etc/system` to reduce the size of the buffer cache is recommended:

```
set bufhwm=28000
```

You can monitor the buffer cache hit rate using `sar -b`. The statistics for the buffer cache show the number of logical reads and writes into the buffer cache, the number of physical reads and writes out of the buffer cache, and the read/write hit ratios.

Try to obtain a read cache hit ratio of 100 percent on systems with a few, but very large files, and a hit ratio of 90 percent or better for systems with many files.

```
# sar -b
SunOS pansco-sdm 5.6 Generic_105181-13 sun4u      07/15/99

00:00:01 bread/s lread/s %rcache bwrit/s lwrit/s %wcache pread/s pwrit/s
01:00:01    0     24     99      3     10     74      5      7
02:00:01    0      7    100      1      6     79      6      2
03:00:01    0      7    100      1      7     83      5      0
04:00:00    0      6     99      1      8     82      5      0
05:00:01    0      6     97      3     12     79      5      0
06:00:01    1      9     91      6     30     79      5      0
07:00:03    1     32     97     12     56     78      4      0
08:00:05    0     37     99     14     57     76      4      0
08:20:02    1     28     98     14     52     74      4      0
08:40:01    3     69     95     15     66     78      3      0
09:00:02   14     31     55     13     55     76      3      0
09:20:02    1     20     97     12     58     79      4      0
09:40:01    0     24     99      9     36     75      4      0
10:00:01    0     16     99      9     41     77      4      0
10:20:01    0     12     98      8     43     81      5      0

Average     1     19     95      7     29     78      5      1
#
```

The best RAM shortage indicator is the scan rate (*sr*) output from *vmstat*. A scan rate above 200 pages per second for long periods (30 seconds) indicates memory shortage. Make sure you have the most recent OS version or, at least, the most recent kernel patch installed.

```
# vmstat 2
procs      memory          page          disk        faults       cpu
r b w    swap  free  mf   pi   po   fr de sr s0 s1 s2 s3 in  sy  cs us sy id
1 1 34 121984 41312 13 2612 11218 2439 14303 0 1528 12 0 0 6 1425 2872 926 15 21 63
0 0 71 6866280 97728 37 4056 22808 11392 44380 0 4220 2 0 0 0 2391 5459 969 22 46 32
0 1 71 6866280 94832 49 5049 28232 12948 47268 0 4347 6 1 0 0 2345 5559 1076 24 53 22
0 2 71 6868128 92984 5 4935 31444 20432 55256 0 4467 13 0 0 0 2364 5530 1056 27 44 29
0 1 71 6868416 95480 5 4621 30732 20100 53460 0 4287 4 0 0 0 2637 5138 1175 24 44 32
^C#
```

Excessive paging is evident by constant nonzero values in the scan rate (*sr*) and page out (*po*) columns. Pageouts are the number of Kbytes paged out per second. As you can see from the output of the *vmstat* command, pageouts are quite large and the scan rate is well over 200 pages per second, indicating a memory shortage.

By looking at the memory requirements of the SAS application (using /usr/proc/bin/pmap -x PID) and looking at the amount of memory allocated to the kernel (sar -k), you have determined that the system does have adequate memory.

Recall that the page daemon begins scanning when the number of free pages is greater than *lotsfree*. The *free* column of the vmstat command reports the size of the free memory list in Kbytes. These are pages of RAM that are immediately ready to be used whenever a process starts up or needs more memory.

Increasing the size of *lotsfree* is appropriate in a system with large amounts of memory. This will reduce the amount of scanning because scanning does not take place until the number of free pages is greater than *lotsfree*. To increasing *lotsfree* to 256 Mbytes is recommended on this system. Make the following entry in the /etc/system file:

```
set lotsfree=0x10000000
```

Many of the sar command options also evaluate memory. sar -g reports the number of pages, per second, scanned by the page daemon (*pgscan/s*). If this value is high, the page daemon is spending a lot of time checking for free memory. This implies that more memory may be needed (or the kernel parameter [*lotsfree*] should be increased).

The number of page out requests per second (*pgout/s*) are also displayed as well as the actual number of pages that are paged out per second (*ppgout/s*). The number of pages per second that are placed on the free list by the page daemon (*pgfree/s*) are also displayed.

```
# sar -g
```

```
SunOS pansco-sdm 5.6 Generic_105181-13 sun4u      07/15/99

00:00:01  pgout/s ppgout/s pgfree/s pgscan/s %ufs_ipf
01:00:01    0.16     0.26   1087.63   1055.39     0.00
02:00:01    0.16     0.27    77.66    83.49     0.00
03:00:01    0.15     0.27    82.27    72.65     0.00
04:00:00    0.27     1.27   134.35   137.28     0.00
05:00:01    0.87     5.98   100.98   84.64     0.00
06:00:01    1.89    11.58   539.67   525.53     0.00
07:00:03   33.47   262.36   2816.19   2584.70     0.00
08:00:05  176.68  1383.28   6086.32   4931.94     0.00
08:20:02  205.35  1614.82   6049.20   4643.43     0.00
08:40:01  188.46  1497.26   5477.24   4238.56     0.00
09:00:02  192.56  1521.26   5507.22   4215.31     0.18
09:20:02  113.15  883.76   4027.47   3246.40     0.00
09:40:01  127.03  1004.41   4296.40   3380.89     0.00
10:00:01   68.85  545.83   3400.88   2861.87     0.00
10:20:01   15.34  120.68   2090.83   1996.26     0.00

Average    50.05   392.96   2052.27   1709.92     0.04
#
```

It is normal to have high paging and scan rates with heavy file system usage as is the case in this customer's environment. However, this can have a very negative effect on application performance. The effect can be noticed as poor interactive response, trashing of the swap disk, and low CPU utilization due to the heavy paging. This happens because the page cache is allowed to grow to the point where it steals memory pages from important applications. Priority paging addresses this problem.

Recall, priority paging is a new paging algorithm which allows main memory to operate as sort of a Harvard cache: providing preferential treatment for instructions (non-file pages) over data (file pages). The priority paging algorithm allows the system to place a boundary around the file cache, so that file system I/O does not cause paging of applications.

By default, priority paging is disabled. In order for the customer to use priority paging, they need either a Solaris 7, Solaris 2.6 with kernel patch 105181-13, or Solaris 2.5.1 with kernel patch 103640-25 or higher environment. This customer is running the Solaris 2.6 OS so the first thing to do is load kernel patch 105181-13 before priority paging can be enabled. Then, to enable priority paging, set the following in the /etc/system file:

```
set priority_paging=1
```

Priority paging introduces a new additional water mark, *cachefree*. The default value for *cachefree* is twice *lotsfree*, or 512 Mbytes for this customer's system.

Case Study – File System Tuning

File system performance is heavily dependent on the nature of the application generating the load. Before configuring a file system, you need to understand the characteristics of the workload that is going to use the file system.

The SAS application is an example of a data-intensive application. Data-intensive applications access very large files. Large amounts of data are moved around without creating or deleting many files.

SAS is essentially a decision support system (DSS). DSSs are characterized by large I/O size (64 KBytes to 1 MByte) and are predominately sequential access.

For data-intensive file systems, the default number of 16 cylinders per cylinder group is too small. Remember, the file system attempts to keep files in a single cylinder group, so make sure that the cylinder groups are big enough. With the large file sizes of a DSS, the number of cylinders per cylinder group should be increased to the maximum of 32.

```
# newfs -c 32 /dev/rdsk/xxx
```

Most file systems implement a read ahead algorithm in which the file system can initiate a read for the next few blocks as it reads the current block. Given the repeating nature of the sequential access patterns of a DSS, the file system should be tuned to increase the number of blocks that are read ahead.

The UFS file system uses the cluster size (*maxcontig* parameter) to determine the number of blocks that are read ahead. This defaults to seven 8-Kbyte blocks (56 Kbytes) in Solaris OS versions up to Solaris 2.6. In the Solaris 2.6 OS, the default changed to the maximum size transfer supported by the underlying device, which defaults to sixteen 8-Kbyte blocks (128 Kbytes) on most storage devices. The default values for read ahead are often too low and should be set very large to allow optimal read rates.

The customer has decided to use RAID-0+1 (striping + mirroring) on the file systems. Sequential reads improve through striping and the customer also wants the high reliability and failure immunity of mirroring. RAID-0+1 combines the concatenation features of RAID-0 with the mirroring features of RAID-1. RAID-0+1 also avoids the problem of writing data sequentially across the volumes, potentially making one drive busy while the others on that side of the mirror are idle.

Cluster size should equal the number of stripe members per mirror multiplied by the stripe size. Two stripe members with a stripe size of 128 Kbytes results in an optimal cluster size of 512 Kbytes. The **-C** option to newfs or the **-a** option to tunefs sets the cluster size.

```
# newfs -C 512 /dev/rdsk/xxx
```

SCSI drivers in a Solaris environment limit the maximum size of an SCSI transfer to 128 Kbytes by default. Even if the file system is configured to issue 512-Kbyte requests, the SCSI drivers will break the requests into smaller 128-Kbyte chunks.

Cluster sizes larger than 128 Kbytes also require the *maxphys* parameter to be set in */etc/system*. The following entry in */etc/system* provides the necessary configuration for larger cluster sizes:

```
set maxphys = 16777216
```

Check Your Progress

Check that you are able to accomplish or answer the following:

- List four major performance bottlenecks
- Describe the characteristics and solutions for each
- Describe several different ways to detect a performance problem
- Identify the type of performance problem that you have, and what tools are available to resolve it

Think Beyond

Things to consider:

- What will you look at first when a performance problem is reported? Why?
- What kind of problems do large applications pose for system tuning?
- How do you know when your system is well tuned?
- What kinds of performance problems might your systems have that will require special monitors or tuning approaches?

Performance Tuning Summary Lab L11

Objectives

Upon completion of this lab, you should be able to:

- Identify the stress on system memory
- Use `vmstat` reports to determine state of the system
- Determine the presence and cause of swapping
- Use `sar` reports to determine system performance

Tasks

vmstat Information

Use the `load11` directory for these exercises.

Complete the following steps using Appendix F as a reference:

1. Type:

```
vmstat -s 5
```

Use this output to answer the following questions. (Run `vmstat` in a command tool window so you can scroll through the output.)

- ▼ How many threads are runnable?

- ▼ How many threads are blocked waiting on disk I/O?

- ▼ How many threads are currently swapped out?

- ▼ How much swap space is available?

- ▼ How much free memory is available?

- ▼ Is the system paging in?

- ▼ Is the system paging out?

- ▼ Where is the system spending most of its time (user mode, kernel mode, or idle)?

2. Type:

```
./load11.sh
```

Note - `load11.sh` may take 3 to 4 minutes to complete.

Note any changes in the vmstat output for the following areas:

- ▼ Pagein activity _____
- ▼ Pageout activity _____
- ▼ Amount of free memory _____
- ▼ Disk activity _____
- ▼ Interrupts _____
- ▼ System calls _____
- ▼ CPU time _____

sar Analysis I

Complete the following steps using Appendix F as a reference. In this part of the lab, use `sar` to read files containing performance data. This data is recorded by `sar` using the `-o` option. The data is recorded in binary format. Each `sar` file was generated using a different benchmark.

1. Type:

```
sar -u -f sar.file1
```

What kind of data is `sar` showing you?

In what mode was the CPU spending most of its time?

Look at the `%wio` field. What does this field represent? Is there a pattern?

2. Type the following command to display the CPU load statistics:

```
sar -q -f sar.file1
```

Look at the size of the run queue as the benchmark progresses. What trend do you see? What does this indicate with respect to the load on the CPU?

Note the amount of time the run queue is occupied (%runocc). Does this support your conclusions from the previous step?

What is the average length of the swap queue? The size of the swap queue indicates the number of LWPs that were swapped out of main memory. What might the size of the swap queue tell you with respect to the demand for physical memory? (If this field is blank, then the swap queue was empty.)

3. Type the following command to format the statistics pertaining to disk I/O:

```
sar -d -f sar.file1
```

Was the disk busy during this benchmark? (Did the benchmark load the I/O subsystem?) Use the %busy field to make an initial determination.

Was the disk busy with many requests? (See the avgque field.) Did the number of requests increase as the benchmark progressed?



As the benchmark progressed, were processes forced to wait longer for their I/O requests to be serviced? (Use `awwait` and `avserv`.) What trend do you notice in these statistics?

Was the I/O subsystem under a load? What was the average size of a request?

4. Type the following command to display memory statistics:

`sar -r -f sar.file1`

What unit is the `freeswap` field in? _____

What was the average value of `freemem`? _____

5. Type the following command to format more memory usage statistics:

`sar -g -f sar.file1`

Was the paging activity steady? (Did the system page at a steady rate?) Explain.

You have just checked three major areas in the SunOS operating system that can affect performance (CPU, I/O, and memory usage). You can use this approach to gather initial data on a suspect performance problem. Given the data you have just collected, what can you conclude with respect to the load on the system?

sar Analysis II

Perform the same analysis as in the previous section on the `sar.file2` file.

1. Type:

```
sar -u -f sar.file2
```

What kind of data is `sar` showing you?

In what mode was the CPU spending most of its time?

Look at the `%wio` field. What does this field represent? Is there a pattern?

2. Type the following command to display the CPU load statistics:

```
sar -q -f sar.file2
```

Look at the size of the run queue as the benchmark progresses. What trend do you see? What does this indicate with respect to the load on the CPU?

Note the amount of time the run queue is occupied (%runocc). Does this support your conclusions from the previous step?

What is the average length of the swap queue? The size of the swap queue indicates the number of LWPs that were swapped out of main memory. What might the size of the swap queue tell you with respect to the demand for physical memory? (If this field is blank, then the swap queue was empty.)

3. Type the following command to format the statistics pertaining to disk I/O:

sar -d -f sar.file2

Was the disk busy during this benchmark? (Did the benchmark load the I/O subsystem?) Use the %busy field to make an initial determination.

Was the disk busy with many requests? (See the avque field.) Did the number of requests increase as the benchmark progressed?

As the benchmark progressed, were processes forced to wait longer for their I/O requests to be serviced? (Use avwait and avserv.) What trend do you notice in these statistics?

Was the I/O subsystem under a load? What was the average size of a request?

4. Type the following command to display memory statistics:

```
sar -r -f sar.file2
```

What unit is the freeswap field in? _____

What was the average value of freemem? _____

5. Type the following command to format more memory usage statistics:

```
sar -g -f sar.file2
```

Was the paging activity steady? (Did the system page at a steady rate?) Explain.

What can you conclude about the load on the system? Which area(s), if any, does this benchmark load?

sar Analysis III

Perform the same analysis as in the previous sections on the sar.file3 file.

1. Type:

```
sar -u -f sar.file3
```

What kind of data is sar showing you?

In what mode was the CPU spending most of its time?

Look at the %wio field. What does this field represent? Is there a pattern?

2. Type the following command to display the CPU load statistics:

```
sar -q -f sar.file3
```

Look at the size of the run queue as the benchmark progresses. What trend do you see? What does this indicate with respect to the load on the CPU?

Note the amount of time the run queue is occupied (%runocc). Does this support your conclusions from the previous step?

What is the average length of the swap queue? The size of the swap queue indicates the number of LWPs that were swapped out of main memory. What might the size of the swap queue tell you with respect to the demand for physical memory? (If this field is blank, then the swap queue was empty.)

3. Type the following command to format the statistics pertaining to disk I/O:

```
sar -d -f sar.file3
```

Was the disk busy during this benchmark? (Did the benchmark load the I/O subsystem?) Use the %busy field to make an initial determination.

Was the disk busy with many requests? (See the avque field.) Did the number of requests increase as the benchmark progressed?

As the benchmark progressed, were processes forced to wait longer for their I/O requests to be serviced? (Use await and avserv.) What trend do you notice in these statistics?

Was the I/O subsystem under a load? What was the average size of a request?



-
4. Type the following command to display memory statistics:

```
sar -r -f sar.file3
```

What unit is the freeswap field in? _____

What was the average value of freemem? _____

5. Type the following command to format more memory usage statistics:

```
sar -g -f sar.file3
```

Was the paging activity steady? (Did the system page at a steady rate?) Explain.

What can you conclude about the load on the system? Which area(s), if any, does this benchmark load?

Interface Card Properties

A 

This appendix provides a list of the performance properties of the system peripheral buses as well as many of the SBus and PCI cards that can be installed in Sun systems.

Additional Resources



Additional resources – The following references can provide additional details on the topics discussed in this appendix:

- Wong, Brian L. 1997. *Configuration and Capacity Planning for Solaris Servers*. Palo Alto: Sun Microsystems Press/Prentice Hall.
- *Sun Field Engineering Handbook*.

SBus Implementations and Capabilities

Platform	Burst Size	Bus Width	Clock	Speed (Read/Write)	Stream Mode
SS1	16 bytes	32 bits	20 MHz	12/20 Mbytes per second	No
SS1+, IPC	16 bytes	32 bits	25 MHz	15/25 Mbytes per second	No
SS2, IPX	32 bytes	32 bits	20 MHz	15/32 Mbytes per second	No
SS10, SS600	32 bytes	32 bits	20 MHz	32/52 Mbytes per second	No
LX, Classic	16 bytes	32 bits	25 MHz	17/27 Mbytes per second	No
SS4, SS5	16 bytes	32 bits	21.25 MHz to 23.3 MHz	36/55 Mbytes per second	No
SS1000, SC2000	64 bytes	32 bits	20 MHz	45/50 Mbytes per second	Yes
SS1000E, SC2000E	64 bytes	32 bits	25 MHz	56/62 Mbytes per second	Yes
SS10SX	128 bytes	32/64 bits	20 MHz	40/95 Mbytes per second	No
SS20	128 bytes	32/64 bits	25 MHz	62/100 Mbytes per second	No
UltraSPARC	128 bytes	32/64 bits	25 MHz	90/120 Mbytes per second	Yes

Typical and Peak Bandwidth of SBus Devices

Board	Description	Typical Throughput	Transfer	Width (Bits)
FSBE/S DSBE/S	Fast SCSI-2/buffered Ethernet SBus fast differential SCSI-2/buffered Ethernet	4 Mbytes per second	DVMA/PIO	32
SunSwift™	Fast/wide SCSI-2 + FastEthernet	12 Mbytes per second	DVMA	64
SOC	Dual 25 Mbytes per second Fibre Channel	16 Mbytes per second	DVMA	64
ATM 1.0	155-Mbit ATM (full duplex)	10 Mbytes per second	DVMA	32
ATM 2.0, 155 Mbit	155-Mbit ATM (full duplex)	10 Mbytes per second	DVMA	64
ATM 2.0, 622 Mbit	622-Mbit ATM (full duplex)	50–100 Mbytes per second	DVMA	64
DWIS/S, SWIS/S	Differential fast/wide Intelligent SCSI-2 host adapter SBus card, fast wide SCSI, single ended	8–10 Mbytes per second	DVMA	32
SBE/S	SCSI, buffered Ethernet, single ended	4 Mbytes per second	DVMA/PIO	32
SCSI	5-Mbytes per second SCSI-2	3 Mbytes per second	DVMA	32
Thick Ethernet	10-Mbits per second Ethernet	500 Kbytes per second	PIO	32
SQEC/S	SBus (card) quad Ethernet controller	3 Mbytes per second	DVMA	32
FastEthernet 1.0	100-Mbits per second FastEthernet	4 Mbytes per second	DVMA	32

Board	Description	Typical Throughput	Transfer	Width (Bits)
FastEthernet 2.0	MII 100-Mbits per second FastEthernet	4 Mbytes per second	DVMA	64
FDDI/S	100-Mbytes per second FDDI	6–8 Mbytes per second	DVMA	32
TRI/S	Token Ring interface (16Mbps) on SBus card	0.8–1.2 Mbytes per second	PIO	32
HSI/S	2x T1 synchronous serial lines	200 Kbytes per second	PIO	32
SPC/S	Serial and parallel controller SBus card	5 Kbytes per second	PIO	32
ISDN	Basic rate ISDN	40 Kbytes per second	PIO	32
SPRN-400	12-ppm + 2 Mbytes per second parallel	2 Mbytes per second	DVMA	32
NP20	20-ppm printer interface	3 Mbytes per second	DVMA	32
VideoPix™	10 fps, 8-bit video interface	6.5 Mbytes per second	PIO	32
SunVideo	30 fps, 24-bit video interface	5–20 Mbytes per second	DVMA	32
MG1, MG2	Monochrome framebuffers	250 Kbytes per second	PIO	32
GX, GX+	8-bit color framebuffers	4–12 Mbytes per second	PIO	32
Turbo GX, TGX+	8-bit color framebuffers	8–20 Mbytes per second	PIO	
GS	Low-cost 24-bit color	15 Mbytes per second	PIO	
GT	24-bit color framebuffer	20 Mbytes per second	PIO/DVMA	64
ZX	Midrange 24-bit color	20–30 Mbytes per second	DVMA	64

Board	Description	Typical Throughput	Transfer	Width (Bits)
SCI	Scalable coherent interface cluster interconnect	10–60 Mbytes per second	PIO/DVMA	64
SOC+ socal	Fibre Channel arbitrated loop	20–70 Mbytes per second	DVMA	64
E&S Freedom 3000	High-performance 24-bit color	40+ Mbytes per second	DVMA	

Installing and Configuring the SyMON System Monitor

B 

This appendix describes how to install and configure the SyMON system monitor for use on your server systems. The procedures in this appendix apply to SyMON releases through 1.6. It does *not* apply to the SyMON 2.0 system monitor.

The Solaris 7 OS does not come with a version of SyMON system monitor. The SyMON 2.0 system monitor supports the Solaris 7 OS, as well as Solaris 2.5.1 and 2.6 OSs. Earlier SyMON versions will not operate on a Solaris 7 OS.

The SyMON 2.0 system monitor can be downloaded from the Sun web site, <http://www.sun.com/symon>.

Additional Resources



Additional resources – The following reference can provide additional details on the topics discussed in this appendix:

- *SyMON Reference and User's Guide.*

SyMON System Monitor

The SyMON system monitor is a distributed server system monitor. It is designed to offer powerful and simple system monitoring capabilities for Sun server systems through a Motif/CDE-based GUI.

Note – The SyMON GUI will work with OpenWindows systems as well. You need to install a Motif runtime package, which is discussed later.

SyMON operation runs three different interacting subsystems:

- The Server subsystem that runs on the monitored server
- The Event Manager/Generator/Handler subsystem that should run on a different system from the server
- The User subsystem that provides the SyMON GUI and related data, which typically runs on the user's desktop system

Only one instance of the Server subsystem and one instance of the Event Generator/Handler can operate per server. Any number of SyMON GUIs can be installed and can operate on different systems to monitor one server.

The SyMON On-Line Diagnostics interface uses SunVTS™. For the On-Line Diagnostics interface to the SunVTS user interface to work, you need to install SunVTS (packages SUNWvts and SUNWt1tk) on both the SyMON user's system and the server system. You also need to start vtsk on the server manually from the `root` account. SunVTS is included in the SMCC Updates to Solaris 2.5.1 Operating Environment CD-ROM and the Software Supplement for the Solaris 2.6 Operating Environment CD-ROM. Additionally, you need to install the SUNWsync package (Solstice SyMON Standalone configd program) on the server system.

The SyMON system monitor uses Tcl scripts extensively to control the layout of certain windows (including any SysMeter layouts you save). These files are saved in your home directory in the `.symon/lib/tcl` directory.

Supported Server Platforms

The 1.6 version of the Solstice SyMON system monitor supports the following platforms:

- SPARCserver™ 1000 and 1000E
- SPARCcenter™ 2000 and 2000E
- Ultra™ Enterprise™ 3000, 4000, 5000, and 6000 servers
- Ultra Enterprise 3500, 4500, 5500, and 6500 servers
- Ultra 1 model 150
- Ultra 2
- Ultra Enterprise 250 and 450

Upgrading the SyMON System Monitor

If you are installing a new version of the SyMON system monitor over a previously installed, older version be sure to do the following:

1. Run `sm_confsymon -D` on all systems involved to avoid version mismatches.
2. Use `pkgrm` to remove all existing SyMON packages before installing the new version.

Installing the SyMON System Monitor

Generally, you should install the server subsystem package on the monitored server, the Event Handler subsystem on another system, and the SyMON User GUI subsystem on the user's desktop system.

While it is permitted, you do not want to install the Event Manager on the monitored system because this will prevent system panics or crashes from being properly registered with the GUI. The Event Handler system can be your desktop system if you want.

The SUNWsyux packages contain the display images for the different supported platforms. Install at least one of these packages on the User GUI subsystem system. Choose the package(s) that contains the images for the system(s) you want to be able to view.

The packages contain the following images:

- SUNWsyua – Ultra Enterprise server family
- SUNWsyub – SPARCserver 1000/E and SPARCcenter 2000/E family
- SUNWsyuc – Ultra Enterprise 2 and Ultra 1 model 150
- SUNWsyud – Ultra Enterprise 450
- SUNWsyue – StorEdgeTM A5000
- SUNWsyuf – Ultra Enterprise 250

Installing the SyMON Packages

Use the pkgadd command to load the appropriate packages into the /opt/SUNWsymon directory.

- On the Server subsystem, enter this command:

```
pkgadd -d release_dir SUNWsyrt SUNWsys
```

- On the Event Manager/Generator/Handler subsystem, enter this command:

```
pkgadd -d release_dir SUNWsyrt SUNWsy
```

- On the GUI User system, enter this command:

```
pkgadd -d release_dir SUNWsyrt SUNWsyu SUNWsyu[abcde]
```

- For man pages on any of the systems, enter this command:

```
pkgadd -d release_dir SUNWsym
```

Environment Variables

Use the following steps to set environment variables in the root user .profile or .login file:

1. Set the *SYMONHOME* environment variable.

```
setenv SYMONHOME /opt/SUNWsymon (csh)  
SYMONHOME=/opt/SUNWsymon; export SYMONHOME (sh, ksh)
```

2. Add \$SYMONHOME/sbin and \$SYMONHOME/bin to your path.

```
setenv PATH $SYMONHOME/sbin:$SYMONHOME/bin:$PATH
```

3. Add \$SYMONHOME/man to your man path.

```
setenv MANPATH $MANPATH:$SYMONHOME/man
```

The following procedures assume that these environment variables have been set correctly.

Configuring the SyMON System Monitor

Configuring the Monitored Server System

As the root user, run the `sm_confsymon` script to set up the SyMON configuration files and to indicate to the Server subsystem which host runs the Event Handler for this server. Then run `sm_control` to activate the monitoring daemons; for example:

```
# sm_confsymon -s ehsystem  
# sm_control start
```

Neither command generates any output. The `sm_confsymon` script also sets up the `/etc/rc2.d/S95symon` script so that the SyMON daemons are started automatically as part of the system boot sequence.

Running the SyMON User Interface

As a typical user, start the GUI monitor for a server as follows:

```
% symon -t server
```

Note that the Server and Event Handler systems must already have the SyMON system monitor installed and running for the user interface to operate properly.

Also, if you are running OpenWindows, you must install the SUNWdtbas package, or the user interface will be unable to start. For the SunOS 5.6 operating system, this package is installed by default. For the SunOS 5.5.1 operating system, it is on the separate Solaris Desktop CD-ROM.

Configuring the Event Handler System

As the root user, run the `sm_confsymon` script to set up the SyMON configuration files and to indicate to the Event Handler subsystem the server for which it will be handling events. Then run `sm_control` to activate the Event Handler daemons; for example:

```
# sm_confsymon -e server
```

You will see the following output when `sm_confsymon` runs:

```
admin# sm_confsymon -e server0
```

Please enter the platform type of server0. SyMON supports:

1. SUNW,SPARCserver-1000
2. SUNW,SPARCserver-1000E
3. SUNW,SPARCcenter-2000
4. SUNW,SPARCcenter-2000E
5. SUNW,Ultra-Enterprise
6. SUNW,Ultra-1 (Only SUNW,Ultra-Enterprise-150
is supported)
7. SUNW,Ultra-2 (Only SUNW,Ultra-Enterprise-2
is supported)
8. SUNW,Ultra-4 (Only SUNW,Ultra-Enterprise-450
is supported)

If you do not know the platform, enter the following command on server0:

```
uname -i
```

Enter platform type or select its number[1-9]. Enter 0 to exit: 5
admin#

```
# sm_control start
```

The `sm_confsymon` script also sets up the `/etc/rc2.d/S95symon` script so that the SyMON daemons are started automatically as part of the system boot sequence.

Setting Up SNMP With Solstice Domain (SunNet) Manager

Complete the following steps:

1. Tell SunNet™ Manager (SNM; or your favorite SNMP agent) about the SyMON traps.

The appropriate SNMP *mib* files are provided with the SyMON system monitor, in the SUNWsyse package, and are installed in the /opt/SUNWsymon/etc/snmp directory.

/opt/SUNWsymon/etc/snmp/symon.mib.oid
/opt/SUNWsymon/etc/snmp/symon.mib.traps
/opt/SUNWsymon/etc/snmp/symon.mib.schema

2. Add the files listed in step 1 to your SNM schema directory. With SNM, copy these files to /opt/SUNWconn/snmp/agents on the SNM host.
3. Load the schema file into your network management application (Console File Load item in SNM).

Directing the SNMP Traps

The component of the SyMON system monitor that sends SNMP traps is the Event Manager. On the Event Manager host, edit the /etc/opt/SUNWsymon/event_gen.*host*.tcl file (where *host* is the monitored host name) and change the set snmp_hosts { } line to include the hostname to which SNMP traps should be sent.

For example, where the host running *snmp* is *wgsnm*, use the following line:

```
set snmp_hosts { wgsnm }
```

Altering the *snmp_hosts* variable can also be done using the sm_confSymon -e event_host -S *host1* command on the Event Host system.

Writing or Modifying Event Rules to Send SNMP Traps

The SyMON `snmp` primitive can be used to send traps from rules. It takes an argument of the string to be sent to the trap. Most default rules already include the `snmp` command, so this will not be necessary in most cases.

How to Stop or Remove SyMON From the System

If you want to kill and deactivate all of the SyMON daemon processes on the Server or Event Handler system, issue the following command:

```
# sm_control stop
```

If you want to remove the SyMON configuration files from your disk as well, issue the following command:

```
# sm_confsymon -D
```

These commands do not remove the data and files installed by pkgadd into /opt/SUNWsymon.

Configuration Issues

Some configuration issues are:

- System resource usage

The `sm_krd` and `sm_egd` agents may require an excessive amount of system resources. This performance problem can be reduced by configuring the system with slower sampling intervals using the `sm_confsymon` command with the `-i n` option.

- SyMON and CDE packages

The SyMON user package requires a package from CDE 1.0.2 or later to be able to run. This package, `SUNWdtbas`, is packaged with the Solaris 2.5.1 software on the Desktop CD-ROM. This CD-ROM includes the CDE packages mentioned previously.

- Colormap limitations

The SyMON system monitor uses about half the colormap with 8-bit graphics. Portions of its graphics (such as icons, images, or event highlighting) might not be displayed if it cannot allocate the required colors. You may have to exit other color-intensive applications (such as Netscape Navigator™) for the SyMON system monitor to operate properly.

- SunVTS interface

To call SunVTS from the SyMON system monitor, you must install these additional packages from the Sun Microsystems Computer Corporation Server Supplements CD-ROM:

- SUNWodu
 - ▼ SUNWsycc
- SUNWvts
 - ▼ SUNWvtssmn

These packages are also on the SMCC Server Supplements CD-ROM.

Accounting

C 

This appendix covers the following:

- Ways in which accounting assists system administrators
- Four types of accounting mechanisms
- The three types of files accounting maintains
- The steps needed to start accounting
- The commands used to generate raw data and the files these commands create
- The commands used to generate reports from the raw data collected

Overview

What Is Accounting?

In the terminal server context, the term *accounting* originally implied the ability to bill users for system services. More recently, the term is used when referring to the monitoring of system operations.

Accounting resources comprise a number of programs and shell scripts that collect data about system use and provide several analysis reports.

What Is Accounting Used For?

Accounting assists system administrators in the following functions:

- Monitoring system usage
- Troubleshooting
- Monitoring system capacity and performance
- Ensuring data security

Types of Accounting

Connection Accounting

Connection accounting includes all information associated with the logging in and out of users. This element of the accounting mechanism does not have to be installed for data to be collected. But if it is not installed, no analysis can be done¹.

When connection accounting is installed, the following data is collected in the `/var/adm/wtmp` file:

- The times at which users logged in
- Shutdowns, reboots, and system crashes
- Terminals and modems used
- The times at which accounting was turned on and off

Process Accounting

As each program terminates, the kernel (the `exit()` function) places an entry in `/var/adm/pacct`. The entry contains the following information:

- The user's user identifier (UID) and group identifier (GID)
- Process start and end times
- CPU time split between user and kernel space
- Amount of memory used
- Number of characters read and written
- Command name (eight characters)
- The process's controlling terminal

1. However, with the `last` command you can print out the `protocol` file.

Disk Accounting

Disk accounting enables you to monitor the amount of data a user has stored on disk. The `dodisk` command available for this purpose should be used once per day. The following information is stored:

- User name and user ID
- Number of data blocks used by a user's files

Charging

The charging mechanisms enable the system administrator to levy charges for specific services (for example, restoring deleted files). These entries are stored in `/var/adm/fee` and are displayed in the accounting analysis report. The following information is stored:

- User name and user ID
- The fee to charge

How Does Accounting Work?

Location of Files

The shell scripts and binaries are located in the `/usr/lib/acct` directory, and the data and report analyses are stored in `/var/adm/acct`.

Types of Files

Accounting maintains three types of files:

- Collection files – Files containing raw data that is appended by the current process or kernel.
- Reports – Data in the form of user reports.
- Summary files – A large number of files containing only summaries. Reports are created from these files.

What Happens at Boot Time

At boot time, the kernel is informed (through the script `/etc/rc2.d/S22acct`) that an entry must be created in `/var/adm/pacct` for each process. The script consists primarily of a call for the following commands:

startup

Starts collection of process information. The data collects in `/var/adm/pacct`, which can grow rapidly, if there is a lot of system activity.

shutacct

Stops data collection and creates an entry in `/var/adm/wtmp`.

Programs That Are Run

Programs started by crontab summarize and analyze the collected data and delete the /var/adm/pacct file after analysis.

- ckpacct – Prevents excessive growth (more than 500 Kbytes) of the /var/adm/pacct file to avoid unnecessarily high compute times during summarizing in the event of an error. In this case, the pacct file is renamed.² The collection of data is continued in a new pacct file. Additionally, accounting is halted when free space drops below 500 Kbytes in /var/adm.
- dodisk – Scans the disk and generates a report on disk space currently in use.
- runacct – Summarizes the raw data collected over the day, deletes the raw data file, and generates analyses as described in the following paragraphs.
- monacct – Generates an overall total from the day's totals, and creates a summary report for the entire accounting period. Additionally, the daily reports are deleted and the summary files reset to zero.

A number of commands are started by runacct and monacct; they can also be started manually.

- lastlogin – Lists all known user names and the date on which these users last logged in
- nulladm – Creates empty files with correct permissions
- prdaily – Generates the daily report from the accounting files

Additionally, the following program is available to the system administrator:

chargefee

If used by the system administrator,
registers additional fees

2. To create the new file name, a digit is added to the old file name. The digit increases; /var/adm/pacct becomes /var/adm/pacct1, /var/adm/pacct2, and so on.

Location of ASCII Report Files

The analysis and summary files are stored in the `/var/adm/acct/sum` and `/var/adm/acct/fiscal` directories.

- `/var/adm/acct/sum` – Contains the daily summary files and daily reports. These files are created by `runacct` and the scripts and programs `runacct` invokes.
- `/var/adm/acct/fiscal` – Contains the monthly analyses and summary files. These files are created by `monacct`.

Starting and Stopping Accounting

Accounting is started using the following steps:

1. Install the SUNWaccr and SUNWaccu packages using the pkgadd or swmtool command.
2. Install the /etc/init.d/acct script as the start script in run level 2:

```
# ln /etc/init.d/acct /etc/rc2.d/S22acct
```

3. Install the /etc/init.d/acct script as the stop script in run level 0:

```
# ln /etc/init.d/acct /etc/rc0.d/K22acct
```

4. Modify the crontabs for users adm and root in order to start the programs dodisk, ckpacct, runacct, and monacct automatically.

```
# EDITOR=vi;export EDITOR
# crontab -e
30 22 * * 4 /usr/lib/acct/dodisk
# crontab -e adm
0 * * * * /usr/lib/acct/ckpacct
30 2 * * * /usr/lib/acct/runacct 2> \
    /var/adm/acct/nite/fd2log
30 7 1 * * /usr/lib/acct/monacct
```

5. Modify the file /etc/acct/holidays, which is used to determine the prime work times. (The reports generated by the accounting programs show a separate total for Prime [prime work time] and Nonprime [other times].)

A line beginning with an asterisk (*) is a comment line. This file must be modified each year to update the system with the changing national or local holidays. An example `holidays` file is shown on the next page.

6. Reboot the system, or type:

```
# /etc/init.d/acct start
```

Following is an example of the /etc/acct/holidays file:

```
# vi /etc/acct/holidays
* @(#)holidays 2.0 of 1/1/99
* Prime/Nonprime Table for UNIX Accounting System
*
* Curr  Prime  Non-Prime
* Year   Start   Start
*
1999 0800    1800
*
* only the first column (month/day) is significant.
*
* month/day      Company
*                      Holiday
*
*
* Attention! Holidays with annually changing dates
* have to be updated each year.

1/1          New Years Day
1/16         Martin Luther King's Birthday
2/20         President's Day
5/29         Memorial Day
7/4          Independence Day
9/4          Labor Day
11/23        Thanksgiving Day
11/24        Day After Thanksgiving
12/25        Christmas Day
```

Generating the Data and Reports

Raw Data

Raw data is stored in four separate files:

- /var/adm/pacct
- /var/adm/wtmp
- /var/adm/acct/nite/disktacct
- /var/adm/fee

/var/adm/pacct File

Following the startup of accounting, the kernel (the `exit()` function) writes an entry in the `/var/adm/pacct` file whenever a process terminates. This is the only logging function that needs to be started explicitly. The definition of the format for entries in this file can be found in `<sys/acct.h>`³ under the structure name `acct`.

/var/adm/wtmp File

The `/var/adm/wtmp` file contains information on logins and logouts and boot operations and shutdowns. The defining structure for the format of this file is called `struct utmp` and is defined in `<utmp.h>`. Entries in the `/var/adm/wtmp` file are written by the following programs:

- `init` – When the system is booted and stopped
- `startup` and `shutacct` – When process accounting is started and stopped
- `ttymon` – The port monitor that monitors the serial port for server requests such as login
- `login` – At login

3. Acute brackets refer to files in the `/usr/include` directory. Therefore, `<sys/acct.h>` means `/usr/include/sys/acct.h`. These structures will be discussed later in this appendix.

/var/adm/acct/nite/disktacct File

Entries are generated once a day by dodisk. The format of the tacct structure is described in the acct(4) man page. It is also illustrated later in this appendix on page -27.

/var/adm/fee File

Only the chargefee command places data in the /var/adm/fee file. The structure consists of the UID, user name, and a number specifying the fee the user pays for a service. This number has a relative rather than an absolute value, which is determined only when the daily and monthly reports are processed.

UID	Username	Fillbytes	Fee
101	otto	0 0 0 0 0 0 0 0 0 0	1500

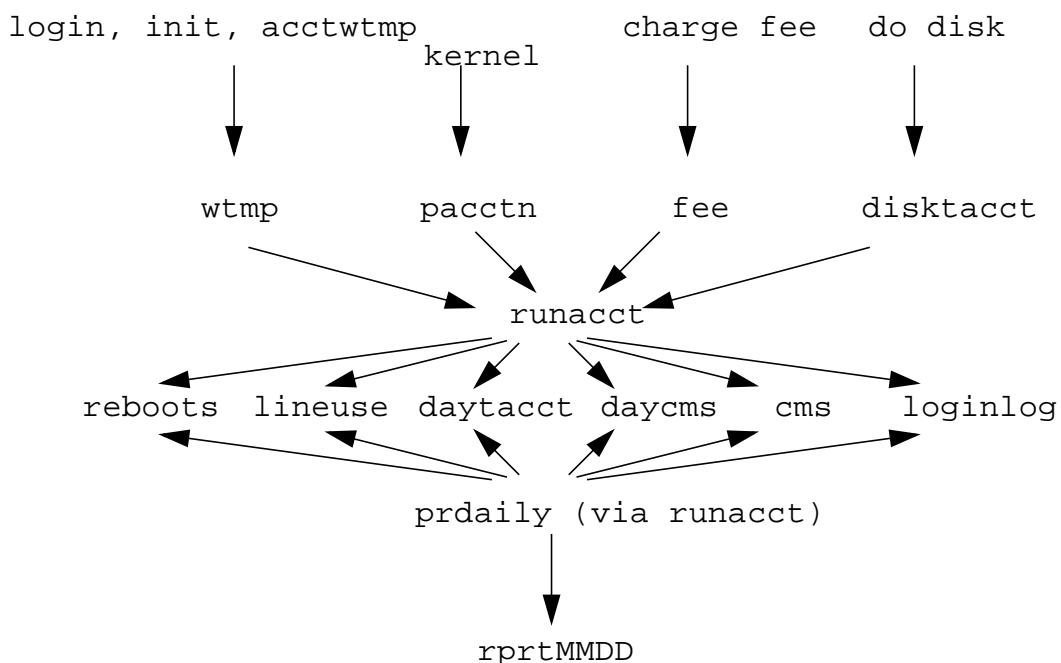
rprtMMDD Daily Report File

A daily report file, `/var/adm/acct/sum/rprtMMDD`, is generated each night by `runacct` from the raw data that has been collected. *MM* represents the month (two digits), and *DD* represents the day.

This file is composed of five parts (subreports): Daily Report, Daily Usage Report, Daily Command Summary, Monthly Total Command Summary, and a Last Login Report. The `runacct` program generates daily summary reports, deletes the raw data, and then calls the script `prdaily` to generate the five subreports in the file `rprtMMDD`.

To bill users, the reports must be processed further as no cost factors are stored in the system (for example, 1 second CPU = \$10). This also applies to charges logged by `chargefee`.

Generation of the rprtMMDD File



Daily Report – Connections

The Daily Report shows the use of the system interfaces and the basic availability.

Generating the Data and Report

```
Apr 16 12:28 1998  DAILY REPORT FOR jobe P
from Mon Dec  1 12:15:15 1997
to   Wed Apr 15 13:57:15 1998
24      system boot
3       run-level 3
3       acctg on
2       run-level 6
2       acctg off
1       acctcon
```

As shown in this output, the following fields are displayed:

- **from** and **to** – The time period to which the report applies.
- **acctcon**, **runacct**, and so on – More specific activities such as booting, shutdown, and starting and stopping of accounting that are stored in `/var/adm/wtmp`.
- **TOTAL DURATION** – The time for which the system was available to the user during this accounting period.
- **LINE** – The interface described by the next line.
- **MINUTES** – Number of minutes the interface was active; that is, when someone was logged in.
- **PERCENT** – The percentage of minutes in the total duration.
- **# SESS** – Number of times login was started by the port monitor.

- # ON – The same number as # SESS as it is no longer possible to invoke login directly.
- # OFF – The number of logouts at which control was returned (voluntarily or involuntarily) to the port monitor. Interrupts at the interface are also shown. If the number is significantly higher than the number of # ONS, a hardware error is probably indicated. The most common cause of this is an unconnected cable dangling from the multiplexer.

Daily Usage Report

The Daily Usage Report shows system usage by users.

Apr 16 12:28 1998 DAILY USAGE REPORT FOR jobe Page 1

```
LOGINCPU(MINS)KCORE-MINCONNECT(MINS)DISK# OF# OF# DISKFEE
UIDNAMEPRIMENPRIMEPRIME NPRIMEPRIMENPRIMEBLOCKSPROCSSESSSAMPLES
OTOTAL6112619614791023848782431500
0root160131514561000976205110
5uucp001200056010
7987otto45124726002322872521311500
```

As shown in this output, the following fields are displayed:

- UID – User identification number.
- LOGIN NAME – User name.
- CPU (MINS) – Number of CPU minutes used. This information is split into PRIME and NPRIME (non-prime), where PRIME is normal work hours and NPRIME is other times such as nights and weekends. The split is determined by the file /etc/acct/holidays, which is set up when accounting is installed.
- KCORE (MINS) – A cumulative measure of the amount of memory in Kbyte segments per minute that a process uses while running. It is divided into PRIME and NPRIME utilization.
- CONNECT (MINS) – The actual time the user was logged in.

- DISK BLOCKS – Number of 512-byte disk blocks at the time `dodisk` was run.
- # OF PROCS – Number of processes invoked by the user. If large numbers appear, a user might have a shell procedure that has run out of control.
- # OF SESS – Number of times the user logged in and out during the reporting period.
- # DISK SAMPLES – Number of times `dodisk` was started to obtain the information in the DISK BLOCKS column.
- FEE – The fees collected by the `chargefee` command (for example, for restoring a file).

Daily Command Summary

This statistic enables the system bottlenecks to be identified. It shows which commands were started and how often, how much CPU time was used, and so on.

Apr 16 12:28 1998 DAILY COMMAND SUMMARY Page 1

```
TOTAL COMMAND SUMMARY
PRIMEPRIMEPRIME
COMMANDNUMBERTOTALTOTALTOTALMEANMEANHOGCHARSBLOCKS
NAME CMDS KCOREMINCPU-MINREAL-MINSIZE-KCPU-MINFACTORTRNSFDREAD
TOTALS78252.86379.5521430.150.140.590.021447776734

sendmail0 8.970.244.8321.460.000.00158130106
grep 74 5.05 0.11 2.50 44.690.000.0550447117

...
```

As shown in this output, the following fields are displayed:

- COMMAND NAME – Name of the command. Unfortunately, all shell procedures are lumped together under the name sh because only object modules are reported by the process accounting system. It is a good idea to monitor the frequency of programs called a.out or core or any other unexpected name. acctcom can be used to determine who executed an oddly named command and if superuser privileges were used.
- NUMBER CMDS – Each command called increments this number by one.
- TOTAL KCOREMIN – Total memory used by these commands in Kbytes per minute.
- PRIME TOTAL CPU-MIN – Total CPU time used by these programs.
- PRIME TOTAL REAL-MIN – Actual elapsed time.
- MEAN SIZE-K – Average memory requirements.
- MEAN CPU-MIN – Average CPU time.
- HOG FACTOR – The ratio of CPU time to actual elapsed time.
- CHARS TRNSFD – Number of characters transferred by read() and write() system calls.
- BLOCKS READ – Number of disk blocks read or written by the program.

Monthly Total Command Summary

The Monthly Total Command Summary report (subreport 4) contains the same fields as the Daily Command Summary report. However, the monthly summary numbers represent accumulated totals since the last execution of monacct.

Last Login

Summary information shows when a user last logged in. It enables unused accounts to be easily identified as the entries are displayed in time sequence.

Apr 16 12:28 1998 LAST LOGIN Page 1

```
00-00-00uucp
00-00-00adm
00-00-00bin
00-00-00sys
95-03-25root
95-04-21otto
95-04-22dmpt
93-10-04guest
92-12-15tjm
92-07-28casey
```

The runacct Program

The main daily accounting shell script, `runacct`, is normally invoked by `cron` outside of prime time hours. The `runacct` shell script processes connect, fee, disk, and process accounting files. It also prepares daily and cumulative summary files for use by `prdaily` and `monacct` for billing purposes.

The `runacct` shell script takes care not to damage files if errors occur. A series of protection mechanisms are used that attempt to recognize an error, provide intelligent diagnostics, and complete processing in such a way that `runacct` can be restarted with minimal intervention. It records its progress by writing descriptive messages into the active

file. (Files used by `runacct` are assumed to be in the `/var/adm/acct/nite` directory, unless otherwise noted.) All diagnostic output during the execution of `runacct` is written into `fd2log`.

When `runacct` is invoked, it creates the files `lock` and `lock1`. These files are used to prevent simultaneous execution of `runacct`. The `runacct` program prints an error message if these files exist when it is invoked. The `lastdate` file contains the month and day `runacct` was last invoked, and is used to prevent more than one execution per day. If `runacct` detects an error, a message is written to the console, mail is sent to `root` and `adm`, locks can be removed, diagnostic files are saved, and execution is ended.

To allow `runacct` to be restartable, processing is broken down into separate re-entrant states. The file `statefile` is used to keep track of the last state completed. When each state is completed, `statefile` is updated to reflect the next state. After processing for the state is complete, `statefile` is read and the next state is processed. When `runacct` reaches the `CLEANUP` state, it removes the locks and ends. States are executed as shown in `runacct` states.

runacct States

SETUP

The command `turnacct` switch is executed to create a new `pacct` file. The process accounting files in `/var/adm/pacctn` (except for the `pacct` file) are moved to `/var/adm/Spacctn.MMDD`. The `/var/adm/wtmp` file is moved to `/var/adm/acct/nite/wtmp.MMDD` (with the current time record added on the end) and a new `/var/adm/wtmp` is created. `closewtmp` and `utmp2wtmp` add records to `wtmp.MMDD` and the new `wtmp` to account for users currently logged in.

wtmpfix

The `wtmpfix` program checks the `wtmp.MMDD` file in the `nite` directory for accuracy. Because some date changes will cause `acctcon` to fail, `wtmpfix` attempts to adjust the time stamps in the `wtmp` file if a record of a date change appears. It also deletes any corrupted entries from the `wtmp` file. The fixed version of `wtmp.MMDD` is written to `tmpwtmp`.

CONNECT

The acctcon program is used to record connect accounting records in the file ctacct.MMDD. These records are in tacct.h format. In addition, acctcon creates the lineuse and reboots files. The reboots file records all the boot records found in the wtmp file.

PROCESS

The acctprc program is used to convert the process accounting files, /var/adm/Spacctn.MMDD, into total accounting records in ptacctn.MMDD. The Spacct and ptacct files are correlated by number so that if runacct fails, the Spacct files will not be processed.

MERGE

The MERGE program merges the process accounting records with the connect accounting records to form daytacct.

FEES

The MERGE program merges ASCII tacct records from the fee file into daytacct.

DISK

If the dodisk procedure has been run, producing the diskacct file, the DISK program merges the file into daytacct and moves diskacct to /tmp/diskacct.MMDD.

MERGETACCT

The MERGETACCT program merges daytacct with sum/tacct, the cumulative total accounting file. Each day, daytacct is saved in sum/tacct.MMDD, so that sum/tacct can be re-created if it is corrupted or lost.

CMS

The `acctcms` program is run several times. `acctcms` is first run to generate the command summary using the `Spacctn` files and write it to `sum/daycms`. The `acctcms` program is then run to merge `sum/daycms` with the cumulative command summary file `sum/cms`. Finally, `acctcms` is run to produce the ASCII command summary files, `nite/daycms` and `nite/cms`, from the `sum/daycms` and `sum/cms` files, respectively. The `lastlogin` program is used to create the `/var/adm/acct/sum/loginlog` log file, the report of when each user last logged in. (If `runacct` is run after midnight, the dates showing the time last logged in by some users will be incorrect by one day.)

USEREXIT

Any installation-dependent (local) accounting program can be included at this point. `runacct` expects it to be called `/usr/lib/acct/runacct.local`.

CLEANUP

This program cleans up temporary files, runs `prdaily` and saves its output in `sum/rpt.MMDD`, removes the locks, then exits. Remember, when restarting `runacct` in the CLEANUP state, remove the last `ptacct` file because it will not be complete.

Periodic Reports

Monthly reports (generated by `monacct`) follow the same formats as the daily reports. Further custom analysis is required to use these reports for true accounting. Certain features should, however, be noted:

- The report includes all activities since `monacct` was last run. There is no interface report.
- The report is in `/var/adm/acct/fiscal/fiscrptMM`, where `MM` represents the current month.
- Monthly summary files are held in `/var/adm/acct/fiscal`.
- Daily summary files are deleted following monthly reporting.
- Daily reports are deleted following monthly reporting.

Looking at the pacct File With acctcom

At any time, you can examine the contents of the /var/adm/pacctn files, or any file with records in the acct.h format, by using the acctcom program. If you do not specify any files and do not provide any standard input when you run this command, acctcom reads the pacct file. Each record read by acctcom represents information about a dead process. (Active processes can be examined by running the ps command.) The default output of acctcom provides the following information:

- Command name (pound [#] sign if it was executed with superuser privileges)
- User
- tty name (listed as ? if unknown)
- Starting time
- Ending time
- Real time (in seconds)
- CPU time (in seconds)
- Mean size (in Kbytes)

The following information can be obtained by using options to acctcom:

- State of the fork/exec flag (1 for fork without exec)
- System exit status
- Hog factor
- Total kcore minutes
- CPU factor
- Characters transferred
- Blocks read

acctcom *Options*

Some options that can be used with acctcom are:

- **-a** – Shows some average statistics about the processes selected. (The statistics are printed after the output is recorded.)
- **-b** – Reads the files backward, showing latest commands first. (This has no effect if reading standard input.)
- **-f** – Prints the fork/exec flag and system exit status columns. (The output is an octal number.)
- **-h** – Instead of mean memory size, shows the hog factor, which is the fraction of total available CPU time consumed by the process during its execution. Hogfactor= $\text{total_CPU_time} \div \text{elapsed_time}$.
- **-i** – Prints columns containing the I/O counts in the output.
- **-k** – Shows total kcore minutes instead of memory size.
- **-m** – Shows mean core size (this is the default).
- **-q** – Does not print output records, just prints average statistics.
- **-r** – Shows CPU factor: $\text{user_time} \div (\text{system_time} + \text{user_time})$.
- **-t** – Shows separate system and user CPU times.
- **-v** – Excludes column headings from the output.
- **-C sec** – Shows only processes with total CPU time (system plus user) exceeding *sec* seconds.
- **-e time** – Shows processes existing at or before *time*, given in the format hr[:min[:sec]].
- **-E time** – Shows processes starting at or before *time*, given in the format hr[:min[:sec]]. Using the same time for both **-S** and **-E**, shows processes that existed at the *time*.
- **-g group** – Shows only processes belonging to *group*.
- **-H factor** – Shows only processes that exceed *factor*, where *factor* is the hog factor (see the **-h** option).
- **-I chars** – Shows only processes transferring more characters than the cutoff number specified by *chars*.

-
- `-l line` – Shows only processes belonging to the terminal `/dev/line`.
 - `-n pattern` – Shows only commands matching *pattern* (a regular expression except that "+" means one or more occurrences).
 - `-o ofile` – Instead of printing the records, copies them in `acct.h` format to *ofile*.
 - `-O sec` – Shows only processes with CPU system time exceeding *sec* seconds.
 - `-s time` – Shows processes existing at or after *time*, given in the format `hr[:min[:sec]]`.
 - `-S time` – Shows processes starting at or after *time*, given in the format `hr[:min[:sec]]`.
 - `-u user` – Shows only processes belonging to *user*.

Generating Custom Analyses

Writing Your Own Programs

You can write your own programs to process the raw data collected by the accounting programs and generate your own reports. For your programs to process the raw data, they must correctly use the format of the entries in the raw data files.

The following pages show the structures that define the format of the entries in the raw data files. Where custom analysis programs are used, `runacct` and `monacct` should not be run.

Raw Data Formats

Different structures define the format of the entries in the following raw data files: `/var/adm/wtmp`, `/var/adm/pacct`, and `/var/adm/acct/nite/disktacct`.

/var/adm/wtmp File

An entry is created for each login and logout in the /var/adm/wtmp file. These entries can be identified from your ut_pid and ut_line entries. The following is an extract from <utmp.h> that defines the format for entries in this file::

```
struct utmp {
    char ut_user[8];           /* User login name */
    char ut_id[4];             /* /etc/inittab id
                                usually line #) */
    char ut_line[12];          /* device name
                                (console, lnx) */
    short ut_pid;              /* short for compat.
                                - process id */
    short ut_type;             /* type of entry */
    struct exit_status ut_exit; /* The exit status of
                                a process */
                                /* marked as DEAD_PROCESS. */
    time_t ut_time;            /* time entry was made */
};

/* Definitions for ut_type */
#define EMPTY          0
#define RUN_LVL        1
#define BOOT_TIME      2
#define OLD_TIME       3
#define NEW_TIME       4
#define INIT_PROCESS   5      /* Process spawned by "init" */
#define LOGIN_PROCESS  6      /* A "getty" process waiting
                                for login */
#define USER_PROCESS   7      /* A user process */
#define DEAD_PROCESS   8
#define ACCOUNTING    9
```

/var/adm/pacct *File*

The /var/adm/pacct file is the process accounting file. The format of the information contained in this file is defined in the <sys/acct.h> file.

The following is an extract from <sys/acct.h>:

```
struct acct
{
    char    ac_flag;           /* Accounting flag */
    char    ac_stat;          /* Exit status */
    uid_t   ac_uid;           /* Accounting user ID */
    gid_t   ac_gid;           /* Accounting group ID */
    dev_t   ac_tty;           /* control typewriter */
    time_t  ac_btime;         /* Beginning time */
    comp_t  ac_utime;         /* acctng user time
                                in clock ticks */
    comp_t  ac_stime;         /* acctng system time
                                in clock ticks */
    comp_t  ac_etime;         /* acctng elapsed time
                                in clock ticks */
    comp_t  ac_mem;           /* memory usage */
    comp_t  ac_io;            /* chars transferred */
    comp_t  ac_rw;            /* blocks read or written */
    char    ac_comm[8];        /* command name */
};
```

/var/adm/acct/nite/disktacct *File*

The /var/adm/acct/nite/disktacct file is in tacct format and is created by dodisk. However, dodisk is relatively inefficient as it processes file systems using find. Conversion to quot and awk should present no problems to an experienced shell-script programmer.

The actual tacct structure is defined by the acctdisk program as follows:

```
/*
 * total accounting (for acct period), also for day
 */

struct tacct {
    uid_t          ta_uid;      /* userid */
    char           ta_name[8];   /* login name */
    float          ta_cpu[2];    /* cum. cpu time, p/np
                                (mins) */
    float          ta_kcore[2];  /* cum kcore-minutes, p/np */
    float          ta_con[2];   /* cum. connect time, p/np,
                                mins */
    float          ta_du;        /* cum. disk usage */
    long           ta_pc;       /* count of processes */
    unsigned short ta_sc;      /* count of login sessions */
    unsigned short ta_dc;      /* count of disk samples */
    unsigned short ta_fee;     /* fee for special services */
};
```

Summary of Accounting Programs and Files

Table C-1 summarizes the accounting scripts, programs, raw data files, and report files discussed in this appendix.

Table C-1 Accounting Commands and Files

Commands	Generated Raw Data Files	Generated ASCII Report Files
init	/var/adm/wtmp	
startup/ shutacct	/var/adm/wtmp	
ttymon	/var/adm/wtmp	
login	/var/adm/wtmp	
kernel(exit() function)	/var/adm/pacct	
dodisk	/var/adm/acct/nite/disk- tacct	
chargefee	/var/adm/fee	
runacct ^a		/var/adm/acct/sum/rprtMMDD
monacct		/var/adm/acct/fiscal/fis- crptMM

a. Actually, runacct calls the prdaily script to create the rprtMMDD file

Objectives

Upon completion of this appendix, you should be able to:

- Briefly describe the functions, features, and benefits of the CacheFS file system
- Define the following terms: *back file system*, *front file system*, *cache directory*, and *write-through cache*
- List the files, commands, and processes used by CacheFS
- Mount NFS using CacheFS

Introduction

Function

CacheFS works with NFS to allow the use of high-speed, high-capacity, local disk drives to store frequently used data from a remote file system. As a result, subsequent access to this data is much quicker than having to access the remote file system each time. CacheFS works similarly with High Sierra File System (HSFS)-based media (such as CD-ROMs).

CacheFS is a file system type (`cachefs`) which is available to the system as a loadable kernel module. It uses the new administration command `cfsadmin` and the existing `mount` and `fsck` commands using the `-F cachefs` option.

Characteristics

CacheFS has these characteristics:

- File data is cached.
- File attributes are cached.
- Directory contents are cached.
- Symbolic links are cached.
- When cache fills, disk resources are reclaimed on a least recently used basis.
- Cache contents are verified using an NFS-like consistency model.

Terminology

Some terms used in connection with CacheFS are:

- Back file system

The back file system is the file system being cached (for example, an NFS or HSFS file system).

- Front file system

The front file system is the cache itself (where the UFS file system cache data resides).

- Cache directory

This local UFS directory holds cached data. A file system may be dedicated to the cache directory, or the cache directory may use a pre-existing file system containing other directories.

- Write-through cache

The write policy is write through (all data is written immediately to the back file system, bypassing the cache). This policy can be changed.

Benefits

The benefits of CacheFS are:

- Improved performance

CacheFS improves “user-perceived” performance due to local caching. Network latencies are improved as the use of CacheFS reduces network traffic.

CacheFS also enhances read performance when used with HSFS (CD-ROM) file systems.

- Increased server capacity

CacheFS supports higher ratios of NFS clients per server (up to twice as many than without CacheFS).

- Increased network capacity

Network segmentation may be deferred as the use of CacheFS reduces network loads and extends network capacity.

- Non-volatile storage

The cached data survives system crashes and reboots (clients can quickly resume work using cached data).

Limitations

CacheFS has the following limitations:

- No caching of `root` file systems.
- No benefit for “write mostly” file systems.
- No benefits for applications using “read once” file systems (for example, `news`).
- No benefit if data is frequently modified on the server.
- Initial reads of cached file systems take slightly longer than without a cache since the data is being copied to the local cache directory on disk (subsequent reads are much faster).

Required Software

CacheFS is part of the `SUNWcsu` and `SUNWcsr` packages installed on every system – no additional software is necessary.

Setting Up CacheFS

Use these steps:

1. Create the cache using `cfsadmin`.
2. Mount the file system with `-F cachefs`.

The same cache may be shared between different mount points.

Creating a Cache

CacheFS caches are created with the `cfsadmin` command.

The `-c` option specifies the cache name.

```
cfsadmin -c cachedirname
```

Options to `cfsadmin` are specified after the `-o` option, in the form `option=value`.

Option	Default	Description
maxblocks	90%	Amount of partition available
minblocks	0%	Minimum use amount
threshblocks	85%	When <i>minblocks</i> begins
maxfiles	90%	Maximum % of inodes available
minfiles	0%	Minimum % of inodes
threshfiles	85%	When <i>minfiles</i> begins

Displaying Information

Use the `-l` option to `cfsadmin` to see the current cache information, including the file system ID.

```
cfsadmin -l
```

Deleting a Cache

To delete a CacheFS cache, as `root` on the client system:

1. Unmount all file systems using the cache.
2. Disconnect the cache from the file system, using the file system ID obtained from the `cfsadmin -l` command.

```
cfsadmin -d ID directory
```

3. Remove any leftover data from the cache if it is used by other mount points or it will be reused.

```
fsck -F cachefs directory
```

Mounting With CacheFS

To mount a NFS file system on a client using CacheFS, use the normal NFS mount command format, specifying a file system type of `cachefs` instead of `nfs`. The `mount` options specify the remainder of the required CacheFS information.

```
mount -F cachefs -o backfstype=nfs,cachedir=cachedirname merlin:/docs /nfsdocs
```

Using an Already Mounted NFS File Systems

If the NFS file system is already mounted, you can add CacheFS support for it without having to unmount the file system. To do this, add the `backpath` option to specify the local (client) mount point, and drop the local mount point name from the `mount` command.

```
mount -F cachefs -o backfstype=nfs,cachedir=cachedirname,backpath=/nfsdocs merlin:/docs
```

Using /etc/vfstab

CacheFS mounts can be placed in `/etc/vfstab` like any other mount request. The "device to fsck" is the cache itself. Remember to include the proper mount options.

```
merlin:/doc /cache/doc /nfsdocs cachefs 2 yes rw,backfstype=nfs,cachedir=cachedirname
```

Using the Automounter

Add the appropriate options to the master map statement:

/home	auto-home	-fstype=cachefs, backfstype=nfs, cachedir=cachedirname
-------	-----------	--

In the master map, this provides defaults for all the individual mounts done under the `/home` directory. The options can also be used on direct maps.

cachefs Operation

While running, CacheFS checks the consistency and status of all files it is caching on a regular basis. File attributes (last modification time) are checked on every file access. This can be adjusted using the standard `actimeout` parameter.

fsck

A cache file system uses `fsck` to check the consistency of the cache. It is run at boot time, against the directory specified in `/etc/vfstab`. It will automatically correct any errors it finds.

If necessary, it can be run manually.

```
fsck -F cachefs cachedirname
```

Writes to Cached Data

By default, writes to a cached file result in the entire file being deleted from the cache. This ensures consistency with other users of the NFS file. This is the default CacheFS behavior, and can be specified by using the `write_around` mount option.

If the files in the mount point will be used only by the system that they are mounted to, the `non_shared` option prevents the deletion of an updated file. Both the cache and NFS server will be updated.



Warning – Do not use the `non_shared` option with shared files. Data corruption could occur.

Managing cachefs

cachefsstat

cachefsstat is used to display statistics about a particular local **cachefs** mount point. This information includes cache hit rates, writes, file creates and consistency checks.

```
cachefsstat [-z] [path ...]
```

If no path is given, statistics for all mount points are listed. The **-z** option reinitializes the counters to zero for the specified (or all) paths.

cachefslog

The **cachefslog** command specifies the location of the **cachefs** statistics log for a given mount point and starts logging, or halts logging altogether.

```
cachefslog [-f logfile] [-h] local_mount_point
```

If neither **-f** nor **-h** is specified, the current state of logging and the logfile are listed. **-f** starts logging and specifies the new logfile, and **-h** terminates logging. To inspect the logfile, use the **cachefsstat** command.

Managing cachefs

cachefswssize

Using the information from the `cachefs` log, which is managed by `cachefslog`, `cachefswssize` calculates the amount of cache disk space required for each mounted file system.

```
cachefswssize logfile

# cachefswssize /var/tmp/cfslog
/home/tlgbs
        end size:          10688k
        high water size:   10704k

/mswre
        end size:          128k
        high water size:   128k

/usr/dist
        end size:          1472k
        high water size:   1472k

total for cache
        initial size:      110960k
        end size:          12288k
        high water size:   12304k
```

cachefspack

The `cachefspack` command provides the ability to control and order the files in the `cachefs` disk cache for better performance. The rules for packing files are given in the `packingrules(4)` man page.

In addition to packing the files, in association with the `filesync(1)` command, the packing list specifies files that the user can synchronize with a server.

CacheFS and CD-ROMs

The CacheFS can be used to cache local CD-ROM drives as well as NFS file systems. The only difference in use is that the `backfstype` is specified as `hsfs`.

```
mount -F cachefs -o  
backfstype=hsfs,cachedir=cachedirname,ro, \  
backpath=/cdrom/... /cdrom/.../doc
```

The CacheFS works properly with Volume Manager, which performs the automatic mounts of CD-ROMs.

Exercise: Using CacheFS

Exercise objective – In this lab you will complete the following tasks:



- Create CacheNFS directory
- Mount NFS resources using CacheFS
- Verify CacheFS benefits
- Display CacheFS directory usage
- Remove CacheFS data and directory

Preparation

For this lab you will need:

- A system installed with the SunOS operating system
- A Solaris 1.x or 2.x NFS server system sharing a read-only directory.

Complete the following steps:

1. Use the `dfshares` command to identify the resources shared by the server.

```
# dfshares server_name
<Observe output>
#
```

2. If no resources are available, perform the following steps:
 - a. Log into the server and edit the `/etc/dfs/dfstab` file to share a file system. For example:

```
share -F nfs -o ro -d "Shared Directory"
/back_fs_path
```

- b. Share the file system resource using the `nfs.server` startup script.

```
# /etc/init.d/nfs.server start
```

3. Locate a file system with at least 5 Mbytes of free disk space.
4. Log into the system as superuser, and invoke the CDE or OpenWindows environment.

Tasks

Complete the following steps:

1. Use the snoop command to observe network activity between your system and the server.

```
# snoop between 'uname -n' server_name  
<Observe output>
```

2. Mount a read-only remote file system from the server.

```
#mount server_name:/back_fs_path /mnt  
#
```

3. Change directory to the mount point and list the directory.

```
# cd /mnt  
# ls  
<Observe output>  
#
```

Does the snoop output reflect remote access file operations? Which types of NFS operations occur?

4. List the mounted directory contents once more.

```
# ls  
<Observe output>  
#
```

Is the snoop output updated with more NFS operations? Why?

5. On a local file system with available disk space, create a cache directory named `cachedir`.

```
# cfsadmin -c /front_fs_path/cachedir  
<Observe output>  
#
```

6. Verify the creation of the cache directory.

```
# cfsadmin -l /front_fs_path/cachedir  
<Observe output>  
#
```

7. Unmount the remote file system.

```
# umount /mnt  
#
```

8. Mount the remote file system using `cachefs`.

```
# mount -F cachefs -o \  
backfstype=nfs, cachedir=/front_fs_path/cachedir \  
server_name:/back_fs_path /mnt
```

9. Begin logging the `cachefs` system with `cachefslog` and confirm that it is logging. Reset the caching statistics.

```
# cachefslog -f /var/tmp/cachelog /mnt  
# cachefslog  
# cachefsstat -z
```

10. Verify that the remote file system is cached.

```
# mount  
<Observe output>  
#
```

-
11. Change the directory to the mount point and list the directory.

```
# cd /mnt  
# ls  
<Observe output>  
#
```

Does the snoop output reflect remote file access operations? Which types of NFS operations occur?

12. List the mounted directory contents once more.

```
# ls  
<Observe output>  
#
```

Is the snoop output updated with more NFS operations? Why not?

13. Edit the /etc/vfstab file to include an entry for the cached file system.

```
server_name:/back_fs_path - /mnt cachefs 9 yes \  
ro,backfstype=nfs, cachedir=/front_fs_path/cachedir
```

14. Unmount the cached file system.

15. Use the mountall command to mount the cached file system using the entry in /etc/vfstab.

```
# mountall -F cachefs /etc/vfstab  
#
```

16. Verify that the remote file system is cached.

```
# mount  
<Observe output>  
#
```

17. Using `cachefsstat` and `cachefswssize`, check the performance of the cached file system, then turn off logging with `cachefslog`.

```
# cachefsstat  
# cachefswssize /var/tmp/cachelog  
# cachefslog -h /mnt
```

18. Unmount the cached file system.

19. Use the `cfsadmin` command to obtain the cache ID for the cached file system.

```
# cfsadmin -l /front_fs_path/cachedir  
<Observe output>  
#
```

The cache ID is listed in the final line of output; for example:

```
cfsadmin: list cache FS information  
. . .  
cache_id
```

20. Use the `cfsadmin` command to delete the CacheFS directory contents for the previously cached file system.

```
# cfsadmin -d cache_id /front_fs_path/cachedir
```

21. Verify that the CacheFS directory is updated.

```
# cfsadmin -l /front_fs_path/cachedir  
<Observe output>  
#
```

22. Use the `cfsadmin` command to remove the CacheFS directory.

```
# cfsadmin -d all /front_fs_path/cachedir  
<Observe output>  
#
```

23. Remove the entry in the `/etc/vfstab` file for the cached file system

IPC Tuneables

E 

This appendix discusses the UNIX System V interprocess communication (IPC) facilities. It also provides the meanings and default values for the configuration parameters for the IPC functionality. These are shared memory, semaphores, and message queues.

Additional Resources



Additional resources – The following references can provide additional details on the topics discussed in this appendix:

- Wong, Brian L. 1997. *Configuration and Capacity Planning for Solaris Servers*. Palo Alto: Sun Microsystems Press/Prentice Hall.
- Cockcroft, Adrian, and Richard Pettit. 1998. *Sun Performance and Tuning*. Palo Alto: Sun Microsystems Press/Prentice Hall.

Overview

The Solaris 2.x OS implements a multithreaded version of the System V IPC facility. The IPC facility is often heavily used by databases, middleware, and client/server tools. IPC is a well understood, portable facility.

How to Determine Current IPC Values

One of the important aspects of the flexibility of the Solaris OS is its dynamic kernel. If a driver or service is not needed, it will not be loaded. When it is first used, it will be loaded by the OS and stay in memory for the life of the boot. This is true for the IPC services. The three dynamically loadable kernel modules for the IPC facilities are:

- shmsys – Shared memory facility
- semsys – Semaphore facility
- msgsys – Message queue facility

IPC status is reported by the `ipcs -a` command, which lists detailed information about the current IPC facility usage. If a service has not been loaded, `ipcs` will return a “facility not in system” message. This means that a program which uses this IPC facility has not been run.

The `ipcs` command will show current IPC facility usage and status but it will not show the tuneable parameter settings. The `sysdef` command will list all of the loaded kernel modules and IPC tuneable parameter settings. However, if a facility has not yet been loaded, these values will be zero. As soon as an IPC facility is loaded, `sysdef` will report its tuneable parameter settings.

An unloaded kernel module can be manually loaded with the `modload` command. Any user can run the `ipcs` and `sysdef` commands, but only the root user may run the `modload` command. `modload` takes the filename of

the module to be loaded as its argument. Typically the IPC modules reside in the `/kernel/sys` directory; for example:

```
modload /kernel/sys/shmsys
```

will load the shared memory facility. Once `shmsys` is loaded, `ipcs` and `sysdef` will show shared memory status and tuneable values.

How to Change IPC Parameters

The IPC tuneable values are set at boot time when the kernel reads the system specification file, `/etc/system`. Even though the IPC facility may not be loaded until later, the tuneables are set at boot time.

The default IPC parameters are set to values which made sense many years ago when IPC was designed. It is highly probable that the default values will not be suitable for modern applications.

To change these values, edit the `/etc/system` file and add the appropriate `set` statements. All of the IPC values are integers; for example:

```
set shmsys:shmmax=2097152
```

There are no specific recommended IPC values because SunOS does not use the IPC facility itself. However, application and database vendors may recommend IPC values. Consult the vendor's installation instructions for recommended IPC values, especially for databases.

After setting the tuneables in `/etc/system`, a reboot is required to allow the kernel to read the new settings.

Tuneable Parameter Limits

Parameter maximum values can be deceiving, since almost all the IPC tuneables are declared as `ints`, which on a SPARC have a maximum value of 2,147,483,647. In many cases, the data item itself must fit into something smaller, such as a 2-byte integer field imbedded in a data structure, which reduces the theoretical maximum significantly.

In any event, the maximum values listed here are strictly a technical limitation based on the data type, and in no way should be construed as something attainable in practice.

It is often impossible to provide a realistic maximum value. Some of the tuneables cause the kernel to use memory for storing static structures. Operating system version, processor architecture, and other kernel modules impact the amount of available kernel memory, making it impossible to define a general absolute maximum. Without enough kernel memory space, the facility may not be loadable. Such an obvious error can be easily rectified by reducing the value of the tuneable.

Shared Memory

Shared memory provides one or more segments of virtual memory that can be shared between programs. Normal shared memory is treated no differently from regular virtual memory: it is pageable, has page table entries for each process that uses it, and keeps separate entries in the translation lookaside buffer (TLB) for each user.

The most common use of shared memory is for DBMS system caches, where high performance is important. To provide better performance for users of shared memory, the Solaris OS provides intimate shared memory (ISM). ISM allows the different processes to share page tables for the shared memory segments, using 4-Mbyte page table entries and, more importantly, these ISM segments are locked in memory. Since ISM is non-pageable, it performs much better. Since it is locked down, there must be enough swap space to support the remainder of the virtual memory required by the system. If there is not enough, normal shared memory will be used.

ISM is available by default, but it is not used unless the calling program specifically requests the use of ISM (using the SHM_SHARE_MMU flag). For better performance on a Solaris 2.6 platform, with patch 105181-05 installed, set enable_grp_ism to one in /etc/system. This is not necessary for the Solaris 7 OS. The use of ISM can be disallowed altogether by setting use_ism in /etc/system to zero.

Table E-1 Shared Memory

Tunable	Type	Default	Maximum	Notes
<code>shmmmax</code>	int	131072	2147482647	The maximum size of a shared memory segment. This is the largest value a program can use in a call to <code>shmget(2)</code> . Setting this tunable very high does not hurt anything, since kernel resources are not allocated based on this value alone.

Table E-1 Shared Memory

Tuneable	Type	Default	Maximum	Notes
<i>shmmmin</i>	int	1	2147482647	The smallest possible size of a shared memory segment. This is the smallest value that a program can use in a call to <code>shmget(2)</code> . Since the default is 1 byte, there should never be any reason to change this; making this <code>int</code> too large could potentially break programs that allocate shared segments smaller than <i>shmmmin</i> .
<i>shmmnmi</i>	int	100	65535	The maximum number of shared memory identifiers that can exist in the system at any point in time. Every shared segment has an identifier associated with it; this is the value that <code>shmget(2)</code> returns. The required number of these is completely dependent on the needs of the application; that is, how many shared segments are needed. Setting this randomly high can cause problems, since the system uses this value during initialization to allocate kernel resources. The kernel memory allocated totals <i>shmmnmi</i> x 112 bytes.
<i>shmseg</i>	int	6	65535	The maximum number of shared memory segments per process. Resources are not allocated based on this value. It is simply a limit that is checked to ensure that a program has not allocated or attached to more than <i>shmseg</i> shared memory segments before allowing another attach to complete. The maximum value should be the current value of <i>shmmnmi</i> , since setting it greater than that will not have any effect.

Semaphores

Semaphores are best described as counters used to control access to shared resources by multiple processes. They are most often used as a locking mechanism to prevent processes from accessing a particular resource while another process is performing operations on it. Semaphores are often dubbed the most difficult to grasp of the three types of System V IPC objects.

The name *semaphore* is actually an old railroad term, referring to the crossroad "arms" that prevent cars from crossing the tracks at intersections. The same can be said about a simple semaphore set. If the semaphore is on (the arms are up), then a resource is available (cars may cross the tracks). However, if the semaphore is off (the arms are down), then resources are not available (the cars must wait).

Semaphores are actually implemented as sets, rather than as single entities. Of course, a given semaphore set might only have one semaphore, as in this railroad analogy.

Table E-2 Semaphores

Tunable	Type	Default	Max	Notes
<i>semmap</i>	int	10	2147482647	The number of entries in the semaphore map. The memory space given to the creation of semaphores is taken from the <i>semmap</i> , which is initialized with a fixed number of map entries based on the value of <i>semmap</i> . <i>semmap</i> should never be larger than <i>semnni</i> . If the number of semaphores per semaphore set used by the application is known (n), then you can use: $\text{semmap} = (((\text{semnni} + n - 1) / n) + 1)$. If <i>semmap</i> is too small for the application, you will see WARNING: rmfree map overflow... messages on the console. Set it higher and reboot.

Table E-2 Semaphores

Tunable	Type	Default	Max	Notes
<i>semnni</i>	int	10	65535	The maximum number of semaphore sets, system-wide, or the number of semaphore identifiers. Every semaphore set in the system has a unique identifier and control structure. During initialization, the system allocates kernel memory for <i>semnni</i> control structures. Each control structure is 84 bytes.
<i>semnns</i>	int	60	65535	The maximum number of semaphores in the system. A semaphore set may have more than one semaphore associated with it, and each semaphore has a <i>sem</i> structure. During initialization the system allocates (<i>semnns</i> x size of (struct <i>sem</i>)) bytes of kernel memory. Each <i>sem</i> structure is only 16 bytes, but they should not be over-specified. This number is usually <i>semnns</i> = <i>semnni</i> x <i>semmsl</i> .
<i>semnnu</i>	int	30	2147482647	The system-wide maximum number of undo structures. This value should be equal to <i>semnni</i> , which provides an undo structure for every semaphore set. Semaphore operations done via <i>semop</i> (2) can be undone if the process that used them terminates for any reason, but an undo structure is required to guarantee this.
<i>semmsl</i>	int	25	2147482647	The maximum number of semaphores per unique ID (per set). As mentioned previously, each semaphore set may have one or more semaphores associated with it. This parameter defines what the maximum number of semaphores is per set.
<i>semopm</i>	int	10	2147482647	The maximum number of individual semaphore operations that can be performed in one <i>semop</i> (2) call.

Table E-2 Semaphores

Tunable	Type	Default	Max	Notes
<i>semume</i>	int	10	2147482647	The maximum number of per-process undo structures. Make sure that the application is setting the SEM_UNDO flag when it gets a semaphore; otherwise, undo structures are not needed. <i>semume</i> should be less than <i>semnnu</i> but sufficient for the application. A reasonable value would be <i>semopm</i> times the average number of processes that will be doing semaphore operations at any point in time.
<i>semusz</i>	int	96	2147482647	Listed as the size in bytes of undo structures, the number of bytes required for the maximum number of configured per-process undo structures. During initialization, it is properly set to (sizeof (undo) + (<i>semume</i> x sizeof (undo))), so it should not be changed.
<i>semvmx</i>	int	32767	65535	The maximum value of a semaphore. Due to the interaction with undo structures (and <i>semaem</i>), this parameter should not exceed the maximum of its default value of 32767 (half of 65535) unless you can guarantee that SEM_UNDO is never being requested.
<i>semaem</i>	short	16384	32767	The maximum adjust-on-exit value. This is a signed, short integer because semaphore operations can increase the value of a semaphore, even though the actual value of a semaphore can never be negative. If <i>semvmx</i> is 65535, <i>semaem</i> would need to be at least 17 bits to represent the range of values possible in a single semaphore operation. In general, <i>semvmx</i> or <i>semaem</i> should not be changed unless you <i>really</i> understand how the application(s) will be using the semaphores. And even then, <i>semvmx</i> should be left at its default value.

Message Queues

System V message queues provide a standardized message-passing system that is used by some large commercial applications.

Table E-3 Message Queues

Tunable	Type	Default	Max	Notes
<i>msgmap</i>	int	100	2147482647	The number of entries in the message allocation map. Similar to <code>semmap</code> in semaphores.
<i>msgmax</i>	int	2048	–	The maximum size of a message.
<i>msgmnb</i>	int	4096	–	The maximum number of bytes on the message queue.
<i>msgmni</i>	int	50	–	The maximum number of message queue identifiers.
<i>msgssz</i>	int	8	–	The maximum message segment size.
<i>msgtql</i>	int	40	–	The number of system message headers.
<i>msgseg</i>	short	1024	32767	The number of message segments.

Performance Monitoring Tools

F≡

This appendix is to be used as a reference for the following tools:

- sar
- vmstat
- iostat
- mpstat
- netstat
- nfsstat



Additional Resources

Additional resources – The following references can provide additional details on the topics discussed in this appendix:

- Cockcroft, Adrian, and Richard Pettit. 1998. *Sun Performance and Tuning*. Palo Alto: Sun Microsystems Press/Prentice Hall.
- Wong, Brian L. 1997. *Configuration and Capacity Planning for Solaris Servers*. Palo Alto: Sun Microsystems Press/Prentice Hall.
- Man pages for the various performance tools (sar, vmstat, iostat, mpstat, netstat, nfsstat.)
- Solaris 7 AnswerBook.

```
sar
```

Display File Access Operation Statistics

```
sar -a
```

```
SunOS grouper 5.6 Generic_105181-03 sun4u      05/24/99
```

```
14:45:15  igin/s namei/s dirbk/s  
14:45:17          0          0          0
```

Where:

- **igin/s** – The number of requests made for inodes that were not in the DNLC.
- **namei/s** – The number of file system path searches per second. If **namei** does not find a directory name in the DNLC, it calls **igin** to get the inode for either a file or directory. Most **igin** calls are the result of DNLC misses.
- **dirbk/s** – The number of directory block reads issued per second.

Display Buffer Activity Statistics

```
sar -b
```

```
SunOS grouper 5.6 Generic_105181-03 sun4u      05/25/99
```

```
08:49:23 bread/s lread/s %rcache bwrit/s lwrit/s %wcache  
pread/s pwrit/s  
08:49:25          0          4         100          0          2         67  
0          0
```

Where:

- **bread/s** – Average number of reads per second submitted to the buffer cache from the disk.
- **lread/s** – Average number of logical reads per second from the buffer cache.

- %rcache – Fraction of logical reads found in the buffer cache (100 percent minus the ratio of bread/s to lread/s).
- bwrit/s – Average number of physical blocks (512 blocks) written from the buffer cache to disk per second.
- lwrit/s – Average number of logical writes to the buffer cache per second.
- %wcache – Fraction of logical writes found in the buffer cache(100 percent minus the ratio of bwrit/s to lwrit/s).
- pread/s – Average number of physical reads per second using character device interfaces.
- pwrit/s – Average number of physical write requests per second using character device interfaces.

Display System Call Statistics

```
sar -c

SunOS grouper 5.6 Generic_105181-03 sun4u      05/25/99

08:55:01 scall/s sread/s swrit/s fork/s exec/s rchar/s
wchar/s
08:55:03          306        27        25      0.00      0.00     4141
3769
```

Where:

- scall/s – All types of system calls per second.
- sread/s – The number of read system calls per second.
- swrit/s – The number of write system calls per second.

-
- `fork/s` – The number of `fork` system calls per second (about 0.5 per second on a four- to six-user system). This number will increase if shell scripts are running.
 - `exec/s` – The number of `exec` system calls per second. If `exec/s` divided by `fork/s` is greater than three, look for inefficient *PATH* variables.
 - `rchar/s` – Characters (bytes) transferred by `read` system calls per second.
 - `wchar/s` – Characters (bytes) transferred by `write` system calls per second.

Display Disk Activity Statistics

sar -d

SunOS grouper 5.6 Generic_105181-03 sun4u 05/25/99						
08:58:48	device	%busy	avque	r+w/s	blk/s/s	await
avserv						
08:58:50	fd0	0	0.0	0	0	0.0
0.0	nfs1	0	0.0	0	0	0.0
0.0	nfs2	0	0.0	0	7	0.5
15.5	nfs3	0	0.0	0	18	0.0
7.6	sd0	6	0.1	4	97	0.0
32.1	sd0,a	2	0.1	1	21	0.0
46.4	sd0,b	4	0.1	3	76	0.0
26.7	sd0,c	0	0.0	0	0	0.0
0.0	sd0,d	0	0.0	0	0	0.0
0.0	sd6	0	0.0	0	0	0.0
0.0						

Where:

- device – Name of the disk device being monitored.
- %busy – Percentage of time the device spent servicing a transfer request.
- avque – The sum of the average wait time plus the average service time.
- r+w/s – Number of read and write transfers to the device per second.
- blk/s/s – Number of 512-byte blocks transferred to the device per second.

- **await** – Average time, in milliseconds, that transfer requests wait idly in the queue (measured only when the queue is occupied).
- **avserv** – Average time, in milliseconds, for a transfer request to be completed by the device (for disks, this includes seek, rotational latency, and data transfer times).

Display Pageout and Memory Freeing Statistics (in Averages)

sar -g

```
SunOS grouper 5.6 Generic_105181-03 sun4u      05/25/99
09:05:20 pgout/s ppgout/s pgfree/s pgscan/s %ufs_ipf
09:05:22     0.00      0.00      0.00      0.00      0.00
```

Where:

- **pgout/s** – The number of pageout requests per second.
- **ppgout/s** – The actual number of pages that are pagedout per second. (A single pageout request may involve pagingout multiple pages.)
- **pgfree/s** – The number of pages per second that are placed on the free list by the page daemon.
- **pgscan/s** – The number of pages per second scanned by the page daemon. If this value is high, the page daemon is spending a lot of time checking for free memory. This implies that more memory may be needed.
- **%ufs_ipf** – The percentage of ufs inodes taken off the free list by iget that had reusable pages associated with them. These pages are flushed and cannot be reclaimed by processes. Thus, this is the percentage of igets with page flushes. A high value indicates that the free list of inodes is pagebound and the number of ufs inodes may need to be increased.

Display Kernel Memory Allocation (KMA) Statistics

```
sar -k
```

```
SunOS grouper 5.6 Generic_105181-03 sun4u      05/25/99  
  
09:10:11 sml_mem    alloc   fail   lg_mem    alloc   fail  
ovsz_alloc   fail  
09:10:13 3301376 1847756      0 6266880 5753576      0  
3661824      0
```

Where:

- **sml_mem** – The amount of memory, in bytes, that the KMA has available in the small memory request pool. (A small request is less than 256 bytes.)
- **alloc** – The amount of memory, in bytes, that the KMA has allocated from its small memory request pool to small memory requests.
- **fail** – The number of requests for small amounts of memory that failed.
- **lg_mem** – The amount of memory, in bytes, that the KMA has available in the large memory request pool. (A large request is from 512 bytes to 4 Kbytes.)
- **alloc** – The amount of memory, in bytes, that the KMA has allocated from its large memory request pool to large memory requests.
- **fail** – The number of failed requests for large amounts of memory.
- **ovsz_alloc** – The amount of memory allocated for oversized requests (those greater than 4 Kbytes). These requests are satisfied by the page allocator; thus, there is no pool.
- **fail** – The number of failed requests for oversized amounts of memory.

Display Interprocess Communication Statistics

sar -m

```
SunOS grouper 5.6 Generic_105181-03 sun4u      05/25/99  
09:13:55    msg/s    sema/s  
09:13:57    0.00     0.00
```

Where:

- msg/s – The number of message operations (sends and receives) per second.
- sema/s – The number of semaphore operations per second.

Display Pagein Statistics

sar -p

```
SunOS grouper 5.6 Generic_105181-03 sun4u      05/25/99  
09:15:56    atch/s   pgin/s  ppgin/s  pflt/s   vflt/s  slck/s  
09:15:58    0.00     0.00     0.00     3.98     3.48     0.00
```

Where:

- atch/s – The number of page faults per second that are satisfied by reclaiming a page currently in memory (attaches per second). Instances of this include reclaiming an invalid page from the free list and sharing a page of text currently being used by another process (for example, two or more processes accessing the same program text).
- pgin/s – The number of times per second that file systems receive pagein requests.
- ppgin/s – The number of pages paged in per second. A single pagein request, such as a soft-lock request (slck/s), or a large block size, may involve paging in multiple pages.

- **pflt/s** – The number of page faults from protection errors. Instances of protection faults are illegal access to a page and copy-on-writes. Generally, this number consists primarily of copy-on-writes.
- **vflt/s** – The number of address translation page faults per second. These are known as validity faults, and occur when a valid process table entry does not exist for a given virtual address.
- **slock/s** – The number of faults per second caused by software lock requests requiring physical I/O. An example of the occurrence of a soft-lock request is the transfer of data from a disk to memory. The system locks the page that is to receive the data, so that it cannot be claimed and used by another process.

Display CPU and Swap Queue Statistics

sar -q

```
SunOS grouper 5.6 Generic_105181-03 sun4u      05/25/99  
09:19:48 runq-sz %runocc swpq-sz %swpocc  
09:19:50          22.0        100
```

Where:

- **runq-sz** – The number of kernel threads in memory waiting for a CPU to run. Typically, this value should be less than two. Consistently higher values mean that the system may be CPU-bound.
- **%runocc** – The percentage of time the dispatch queues are occupied.
- **swpq-sz** – The average number of swapped-out LWP).
- **%swpocc** – The percentage of time LWPs are swapped out.

Display Available Memory Pages and Swap Space Blocks

sar -r

```
SunOS rafael 5.7 Generic sun4u      06/11/99  
  
00:00:00 freemem freeswap  
01:00:00      535    397440  
02:00:00      565    397538
```

Where:

- **freemem** – The average number of memory pages available to user processes over the intervals sampled by the command. Page size is machine dependent.
- **freeswap** – The number of 512-byte disk blocks available for page swapping.

Display CPU Utilization (the Default)

sar -u

```
SunOS grouper 5.6 Generic_105181-03 sun4u      05/25/99  
  
09:32:28      %usr      %sys      %wio      %idle  
09:32:30          0          2          0         98
```

Where:

- **%usr** – The percentage of time that the processor is in user mode.
- **%sys** – The percentage of time that the processor is in system mode.
- **%wio** – The percentage of time the processor is idle and waiting for I/O completion.
- **%idle** – The percentage of time the processor is idle and is not waiting for I/O.

Display Process, Inode, File, and Shared Memory Table Sizes

sar -v

```
SunOS grouper 5.6 Generic_105181-03 sun4u      05/25/99  
09:35:19  proc-sz      ov  inod-sz      ov  file-sz      ov  lock-  
sz  
09:35:21      77/1962      0 5178/8656      0 490/490      0  0/0
```

Where:

- **proc-sz** – The number of process entries currently being used, and the process table size allocated in the kernel.
- **inod-sz** – The total number of inodes in memory versus the maximum number of inodes allocated in the kernel. This is not a strict high-water mark; it can overflow.
- **file-sz** – The number of entries and the size of the open system file table.
- **lock-sz** – The number of shared memory record table entries currently being used or allocated in the kernel. The sz is given as 0 because space is allocated dynamically for the shared memory record table.
- **ov** – Overflows that occur between sampling points for each table.

Display Swapping and Switching Statistics

sar -w

```
SunOS grouper 5.6 Generic_105181-03 sun4u      05/25/99  
09:39:03  swpin/s bswin/s swpot/s bswot/s pswch/s  
09:39:05      0.00      0.0      0.00      0.0      118
```

Where:

- **swpin/s** – The number of LWP transfers into memory per second.
- **bswin/s** – The number of blocks (512-byte units) transferred for swap-ins per second.
- **swpot/s** – The average number of processes swapped out of memory per second. If the number is greater than 1, you may need to increase memory.
- **bswot/s** – The number of blocks transferred for swap-outs per second.
- **pswch/s** – The number of kernel thread switches per second.

Display Monitor Terminal Device Statistics

sar -y

```
SunOS grouper 5.6 Generic_105181-03 sun4u      05/25/99  
10:28:10 rawch/s canch/s outch/s rcvin/s xmtin/s mdmin/s  
10:28:12      0        0       58        0        0        0
```

Where:

- **rawch/s** – The number of input characters per second.
- **canch/s** – The number of input characters processed by canon (canonical queue) per second.
- **outch/s** – The number of output characters per second.
- **rcvin/s** – The number of receiver hardware interrupts per second.
- **xmtin/s** – The number of transmitter hardware interrupts per second.
- **mdmin/s** – The number of modem interrupts per second.

Display All Statistics

sar -A

```

SunOS rafael 5.7 Generic sun4u      06/07/99

08:32:41      %usr      %sys      %wio      %idle
                device          %busy    avque   r+w/s   blks/s   aawait
avserv
                runq-sz  %runocc  swpq-sz  %swpocc
                bread/s lread/s %rcache bwrit/s lwrit/s %wcache pread/s
pwrite/s
                swpin/s bswin/s swpot/s bswot/s pswch/s
                scall/s sread/s swrit/s fork/s exec/s rchar/s wchar/s
                iget/s namei/s dirbk/s
                rawch/s canch/s outch/s rcvin/s xmtin/s mdmin/s
                proc-sz   ov  inod-sz   ov  file-sz   ov  lock-sz
                msg/s  sema/s
                atch/s pgin/s ppgin/s pfilt/s vflt/s slck/s
                pgout/s ppgout/s pgfree/s pgscan/s %ufs_ipf
                freemem freeswap
                sml_mem alloc fail lg_mem alloc fail ovsz_alloc
fail

08:32:43      7       2       0      91
                fd0        0       0.0      0       0       0.0
0.0
                nfs1        0       0.0      0       0       0.0
0.0
                nfs2        4       0.1      2     160       0.3
26.0
                nfs3        0       0.0      0       0       0.0
0.0
                nfs5        0       0.0      0       0       0.0
0.0
                sd0         0       0.0      0       0       0.0
0.0
                sd0,a       0       0.0      0       0       0.0
0.0
                sd0,b       0       0.0      0       0       0.0
0.0
                sd0,c       0       0.0      0       0       0.0
0.0
                sd0,d       0       0.0      0       0       0.0
0.0
                sd6         0       0.0      0       0       0.0
0.0

```

	0	0	100	0	0	100	0
0	0.00	0.0	0.00	0.0	242		
	669	49	40	0.00	0.00	213992	92603
	0	8	0				
	0	0	394	0	0	0	0
	59/922	0	2188/4236	0	462/462	0	0/0
	0.00	0.00					
	0.00	0.00	0.00	2.49	3.48	0.00	
	2.49	9.95	10.95	8.46	0.00		
	118	423690					
	2187264	1755488	0	7766016	6979056	0	1990656
0							

Other Options

-e *time*

Select data up to time. Default is 18:00.

-e *filename*

Use filename as the data source. Default is the current daily data file /var/adm/sa/sadd (where dd are digits representing the day of the month).

-i *sec*

Select data at intervals as close as possible to sec seconds.

-o *filename*

Save output in filename in binary format.

-s *time*

Select data later than time in the form hh[:mm]. Default is 08:00.

vmstat

Display Virtual Memory Activity Summarized at Some Optional, User-Supplied Interval (in Seconds)

```
vmstat 2
procs      memory          page          disk          faults
cpu
  r b w    swap   free   re   mf pi po fr de sr aa dd f0 s1   in   sy   cs us
sy id
  0 0 5   99256 73832   0    2   0   1   1   0   1   0   0   0   0   0   113   96   53   0
  0 100
  0 0 21  141688 2000   0    1   0   0   16   0   8   0   0   0   0   0   143  214  95   0
  0 100
  0 0 21  141688 2000   0    0   0   4   4   0   0   0   0   0   0   0   168  264 101   0
  0 99
  0 0 21  141688 2008   0    0   0   0   0   0   0   0   0   0   0   0   138  212  96   0
  0 100
  0 0 21  141688 2008   0    0   0   0   0   0   0   0   0   0   0   0   137  196  89   0
  0 100
```

Where:

- procs – The number of processes in each of the three following states:
 - r – The number of kernel threads in the dispatch (run) queue. The number of runnable processes that are not running.
 - b – The number of blocked kernel threads waiting for resources, I/O, paging, and so forth. The number of processes waiting for a block device transfer (disk I/O) to complete.
 - w – The number of swapped out LWPs waiting for processing resources to finish; the swap queue length. The number of idle swapped-out processes, not explained properly in man pages since that measure used to be the number of active swapped-out processes waiting to be swapped back in.
- memory – Usage of real and virtual memory.

- swap – The amount of available swap space currently available in Kbytes. If this number ever goes to zero, your system will hang and be unable to start more processes.
- free – The size of the free memory list in Kbytes. The pages of RAM that are immediately ready to be used whenever a process starts up or needs more memory.
- page – Page faults and paging activity, in units per second.
 - re – The number of pages reclaimed from the free list. The page had been stolen from a process but had not yet been reused by a different process so it can be reclaimed by the process, thus avoiding a full page fault that would need I/O.
 - mf – The number of minor faults in pages. A minor fault is caused by an address space or hardware address translation fault that can be resolved without performing a pagein. It is fast and uses only a little CPU time. Thus the process does not have to stop and wait, as long as a page is available for use from the free list.
 - pi – The number of Kbytes paged in per second.
 - po – The number of Kbytes paged out per second.
 - fr – The number of Kbytes freed per second. Pages freed is the rate at which memory is being put onto the free list by the page scanner daemon. Pages are usually freed by the page scanner daemon, but other mechanisms, such as processes exiting, also free pages.
 - de – The deficit or the anticipated, short-term memory shortfall needed by recently swapped-in processes. If the value is nonzero, then memory was recently being consumed quickly, and extra free memory will be reclaimed in anticipation that it will be needed.

- **sr** – The number of pages scanned by page daemon as it looks for pages to steal from processes that are not using them often. This number is the key memory shortage indicator if it stays above 200 pages per second for long periods. This is the most important value reported by `vmstat`.
- **disk** – The number of disk operations per second. There are slots for up to four disks, labeled with a single letter and number. The letter indicates the type of disk (s = SCSI, i = IPI, and so forth); the number is the logical unit number.
- **faults** – The trap/interrupt rates.
 - **in** – The number of interrupts per second
 - **sy** – The number of system calls per second
 - **cs** – The number of CPU context switches per second
- **cpu** – A breakdown of percentage usage of CPU time. On multiprocessor systems, this is an average across all processors.
 - **us** – The percentage of time that the processor is in user mode.
 - **sy** – The percentage of time that the processor is in system mode.
 - **id** – The percentage of time that the processor is idle.

Display Cache Flushing Statistics

```
vmstat -c
flush statistics: (totals)
  usr    ctx    rgn    seg    pag    par
  821    960     0     0 123835     97
```

Where:

- **usr** – The number of user cache flushes since boot time.
- **ctx** – The number of context cache flushes since boot time.
- **rgn** – The number of region cache flushes since boot time.
- **seg** – The number of segment cache flushes since boot time.
- **pag** – The number of page cache flushes since boot time.
- **par** – The number of partial-page cache flushes since boot time.

Display Interrupts Per Device

```
vmstat -i
interrupt      total      rate
-----
clock        7734432      100
zsc0         218486       2
zscl         422997       5
hmec0        127107       1
fdc0          10           0
-----
Total        8503032      109
```

Display System Event Information, Counted Since Boot

```
vmstat -s
    40 swap ins
    37 swap outs
    40 pages swapped in
   26071 pages swapped out
 1007708 total address trans. faults taken
    20024 page ins
   12205 page outs
   27901 pages paged in
   52838 pages paged out
    6847 total reclaims
    5841 reclaims from free list
        0 micro (hat) faults
 1007708 minor (as) faults
    19106 major faults
  208321 copy-on-write faults
 197883 zero fill page faults
 640512 pages examined by the clock daemon
        41 revolutions of the clock hand
   65328 pages freed by the clock daemon
    9791 forks
    3022 vforks
    15138 execs
18611577 cpu context switches
 74053744 device interrupts
 1245736 traps
 33533770 system calls
 2000912 total name lookups (cache hits 96%)
        188 toolong
    73340 user     cpu
    58410 system   cpu
 34399437 idle     cpu
    82829 wait     cpu
```

Where:

- **micro (hat) faults** – The number of hardware address translation faults.
- **minor (as) faults** – The number of address space faults.
- **total name lookups** – The DNLC hit rate.

Display Swapping Activity Summarized at Some Optional, User-Supplied Interval (in Seconds)

This option reports on swapping activity rather than the default, paging activity. This option will change two fields in vmstat's "paging" display; rather than the `re` and `mf` fields, `vmstat -S` will report `si` and `so`.

```
vmstat -S 2
procs      memory          page          disk          faults
cpu
  r b w    swap   free   si   so pi po fr de sr aa dd f0 s1   in   sy   cs us
sy id
  0 0 5    104 73704   0   0   0   1   1   0   1   0   0   0   0   113   97   53   0
0 100
  0 0 21   141024 2072   0   0   0   0   0   0   0   0   0   0   0   0   141  226   94   0
0 100
  0 0 21   141024 2072   0   0   0   0   0   0   0   0   0   0   0   0   139  415   98   1
2 96
  0 0 21   141008 2056   0   0   0   4   4   0   0   0   0   0   0   0   134  464  108   3
6 91
  0 0 21   140992 2048   0   0   0   0   0   0   0   0   0   0   0   0   138  201   93   0
0 100
```

Where:

- `si` – The average number of LWPs swapped in per second. The number of Kbytes/sec swapped in.
- `so` – The number of whole processes swapped out. The number of Kbytes/sec swapped out.

iostat

Display CPU Statistics

```
iostat -c  
      cpu  
us sy wt id  
35  9  1 55
```

Where:

- us – Percentage of time the system has spent in user mode.
- sy – Percentage of time the system has spent in system mode.
- wt – Percentage of time the system has spent waiting for I/O.
- id – Percentage of time the system has spent idle.

Display Disk Data Transferred and Service Times

```
iostat -d  
      fd0          sd0          sd6          nfs1  
kps tps serv  kps tps serv  kps tps serv  kps tps serv  
 0   0   0    64   1   57    0   0   0    0   0   0
```

For each disk:

- kps – The number of Kbytes transferred per second.
- tps – The number of transfers per second.
- serv – The average service time in milliseconds.

Display Disk Operations Statistics

```
iostat -D
      fd0          sd0          sd6          nfs1
rps wps util   rps wps util   rps wps util   rps wps util
 0   0  0.0     4   0  6.9     0   0  0.0     0   0  0.0
```

For each disk:

- rps – The number of reads per second.
- wps – The number of writes per second.
- util – The percentage of disk utilization.

Display Device Error Summary Statistics

```
iostat -e
      ----- errors -----
device    s/w h/w trn tot
fd0       0   0   0   0
sd0       0   0   0   0
sd6       0   1   0   1
nfs1      0   0   0   0
nfs2      0   0   0   0
nfs3      0   0   0   0
```

Where:

- s/w – The number of soft errors.
- h/w – The number of hard errors.
- trn – The number of transport errors.
- tot – The total number of all types of errors.

Display All Device Errors

iostat -E

```
sd0      Soft Errors: 0 Hard Errors: 0 Transport Errors: 0
Vendor: SEAGATE Product: ST32430W SUN2.1G Revision: 0508
Serial No: 00155271
Size: 2.13GB <2127708160 bytes>
Media Error: 0 Device Not Ready: 0 No Device: 0 Recoverable: 0
Illegal Request: 0 Predictive Failure Analysis: 0

sd6      Soft Errors: 0 Hard Errors: 1 Transport Errors: 0
Vendor: TOSHIBA Product: XM-5401TASUN4XCD Revision: 1036
Serial No: 04/12/95
Size: 18446744073.71GB <-1 bytes>
Media Error: 0 Device Not Ready: 0 No Device: 1 Recoverable: 0
Illegal Request: 0 Predictive Failure Analysis: 0
```

Display Counts Rather Than Rates, Where Possible

iostat -I

tty	fd0	sd0	sd6	nfs1
cpu				
tin tout kpi tpi serv	kpi tpi serv	kpi tpi serv	kpi tpi serv	
us sy wt id				
0 80 0 0	0 64 8 59	0 0 0 0	0 0 0 0	0 0 0 0
0 0 0 100				

Report on Only the First n Disks

iostat -l 3

tty	fd0	sd0	sd6	cpu
tin tout kps tps serv	kps tps serv	kps tps serv	us sy wt id	
0 33 0 0 0	48 6 92 0 0	0 0 0 0 0	0 0 0 0 0	0 0 0 0 0
100				

Display Data Throughput in Mbytes/sec Rather Than Kbytes/sec

```
iostat -M
      tty          fd0          sd0          sd6          nfs1
cpu
tin tout Mps tps serv  Mps tps serv  Mps tps serv  Mps tps serv
us sy wt id
    0  40   0   0   0   7  77   0   0   0   0   0   0   0
0   0   2  98
```

Display Device Names as Mount Points or Device Addresses

```
iostat -n
      tty          fd0        c0t0d0        c0t6d0        rafael:v
cpu
tin tout kps tps serv  kps tps serv  kps tps serv  kps tps serv
us sy wt id
    0  40   0   0   0   68   9  121   0   0   0   0   0   0
0   0   2  98
```

Display Per-Partition Statistics as Well as Per-Device Statistics

```
iostat -p
      tty          fd0          sd0        sd0,a        sd0,b
cpu
tin tout kps tps serv  kps tps serv  kps tps serv  kps tps serv
us sy wt id
    0 118   0   0   0   53   8   86   53   8   86   0   0   0
0   1   1  98
```

Display Per-Partition Statistics Only

```
iostat -P
    tty      sd0,a      sd0,b      sd0,c      sd0,d
cpu
tin tout kps tps serv  kps tps serv  kps tps serv  kps tps serv
us sy wt id
    0   40  48    7   90    0   0    0   0    0   0    0   0    0
0   0   1 98
```

Display Characters Read and Written to Terminals

```
iostat -t
    tty
tin tout
    0   16
```

Where:

- tin – The number of characters in the terminal input queue.
- tout – The number of characters in the terminal output queue.

Display Extended Disk Statistics in Tabular Form

iostat -x

device	extended device statistics								
	r/s	w/s	kr/s	kw/s	wait	actv	svc_t	%w	%b
fd0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0	0
sd0	18.5	2.8	180.1	122.6	0.0	1.1	52.8	0	25
sd6	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0	0
nfs1	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0	0
nfs2	0.4	0.5	6.4	9.3	0.0	0.0	22.7	0	2
nfs3	2.0	0.0	43.3	0.0	0.0	0.0	7.8	0	1
nfs5	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0	0

Where:

- r/s – The number of reads per second.
- w/s – The number of writes per second.
- kr/s – The number of Kbytes read per second.
- kw/s – The number of Kbytes written per second.
- wait – The average number of transactions waiting for service (queue length). In the device driver, a read or write command is issued to the device driver and sits in the wait queue until the SCSI bus and disk are both ready.
- actv – The average number of transactions actively being serviced (active queue). The average number of commands actively being processed by the drive.
- svc_t – The average I/O service time. Normally, this is the time taken to process the request once it has reached the front of the queue, however, in this case, it is actually the response time of the disk.
- %w – The percentage of time the queue is not empty. The percentage of time there are transactions waiting for service.
- %b – The percentage of time the disk is busy, transactions in progress, commands active on the drive.

Break Down Service Time, svc_t, Into Wait and Active Times and Put Device Name at the End of the Line

```
iostat -xn
                         extended device statistics
      r/s   w/s    kr/s    kw/s wait  actv wsvc_t asvc_t  %w  %b device
      0.0   0.0     0.0     0.0  0.0   0.0    0.0    0.0    0   0 fd0
    10.8   2.2   97.2   68.7  0.0   0.7    0.1   57.1    0  15 c0t0d0
      0.0   0.0     0.0     0.0  0.0   0.0    0.0    0.0    0   0 c0t6d0
      0.0   0.0     0.0     0.0  0.0   0.0    0.0    0.0    0   0
rafael:vold(pid223)
      0.2   0.5     3.7     9.7  0.0   0.0    0.3   21.8    0   1
ra:/export/home13/bob
      1.0   0.0   20.6    0.0  0.0   0.0    0.2    7.4    0   1
shasta:/usr/dist
```

mpstat

```
# mpstat 2
CPU minf mjf xcal    intr ithr    csw icsw migr smtx    srw syscl   usr sys  wt
idl
 0     0     0     0     0     0     35     0     2     1     0     96     0     0     0
100
 2     0     0     4    208     8     34     0     2     1     0     81     0     0     0
100
CPU minf mjf xcal    intr ithr    csw icsw migr smtx    srw syscl   usr sys  wt
idl
 0     2     0     0     0     0     30     0     1     0     0     54     0     0     0
100
 2     0     0     4    212    11     21     0     1     0     0     25     0     0     0
100
CPU minf mjf xcal    intr ithr    csw icsw migr smtx    srw syscl   usr sys  wt
idl
 0     0     0     0     0     0     31     0     1     0     0     53     0     0     2
98
 2     0     0     4    248    48     28     0     1     0     0     35     0     0     0
100
^C#
```

Where:

- CPU – The processor ID.
- minf – The number of minor faults.
- mjf – The number of major faults.
- xcal – The number of inter-processor cross-calls. These occur when one CPU wakes up another CPU by interrupting it.
- intr – The number of interrupts.
- ithr – The number of interrupts as threads (not counting clock interrupt).
- csw – The number of context switches.
- icsw – The number of involuntary context switches.
- migr – The number of thread migrations to another processor.
- smtx – The number of spins on mutexes (mutual exclusion lock). The number of times the CPU failed to obtain a mutex on the first try.

- `srw` – The number of spins on readers/writer locks; that is, the number of times the lock was not acquired on first try.
- `syscl` – The number of system calls.
- `usr` – The percentage of time spent in user mode.
- `sys` – The percentage of time spent in system mode.
- `wt` – The percentage of time spent waiting.
- `idl` – The percentage of time spent idle.

netstat

Show the State of All Sockets

```
# netstat -a

UDP
  Local Address          Remote Address      State
  -----
    *.sunrpc
    *.*
    *.32771
    *.32773
    *.lockd
    ...
    ...
TCP
  Local Address          Remote Address      Swind Send-Q Rwind Recv-Q
  State
  -----
    *.*                  *.*                0      0      0      0
IDLE
    *.sunrpc            *.*                0      0      0      0
LISTEN
    *.*                 *.*                0      0      0      0
IDLE
    *.32771             *.*                0      0      0      0
LISTEN
    *.32772             *.*                0      0      0      0
LISTEN
    *.ftp               *.*                0      0      0      0
LISTEN
    *.telnet            *.*                0      0      0      0
LISTEN
Active UNIX domain sockets
Address  Type      Vnode      Conn  Local Addr      Remote Addr
3000070d988 stream-ord 30000a67708 00000000 /var/tmp/aaa3daGZa
3000070db20 stream-ord 300000fe7b0 00000000 /tmp/.X11-unix/X0
3000070dc8 stream-ord 00000000 00000000
#
```

The display for each active socket shows the local and remote addresses, the send, Send-Q, and receive, Recv-Q, queue sizes (in bytes); the send, Swind, and receive, Rwind, windows (in bytes); and the internal state of the protocol.

The symbolic format normally used to display socket addresses is either hostname.port when the name of the host is specified, or network.port if a socket address specifies a network but no specific host. Unspecified, or wildcard, addresses and ports appear as "*".

The possible state values for TCP sockets are as follows:

- `BOUND` – Bound, ready to connect or listen.
- `CLOSED` – Closed. The socket is not being used.
- `CLOSING` – Closed, then remote shutdown; awaiting acknowledgment.
- `CLOSE_WAIT` – Remote shutdown; waiting for the socket to close.
- `ESTABLISHED` – Connection has been established.
- `FIN_WAIT_1` – Socket closed; shutting down connection.
- `FIN_WAIT_2` – Socket closed; waiting for shutdown from remote.
- `IDLE` – Idle, opened but not bound.
- `LAST_ACK` – Remote shutdown, then closed; awaiting acknowledgment.
- `LISTEN` – Listening for incoming connections.
- `SYN_RECEIVED` – Initial synchronization of the connection underway.
- `SYN_SENT` – Actively trying to establish connection.
- `TIME_WAIT` – Wait after close for remote shutdown retransmission.

Show All Interfaces Under Dynamic Host Configuration Protocol (DHCP) Control

```
# netstat -d
msg 1: group = 263 mib_id = 0 length = 16
msg 2: group = 263 mib_id = 5 length = 980
...
...
--- Entry 1 ---
Group = 263, mib_id = 0, length = 16, valp = 0x2cc60

--- Entry 2 ---
Group = 263, mib_id = 5, length = 980, valp = 0x2da50
49 records for udpEntryTable:
...
...
TCP
  Local Address          Remote Address      Swind Send-Q Rwind Recv-Q
  State
  -----
  -----
rafael.1023           ra.nfsd            8760    0 8760    0
ESTABLISHED
localhost.32805       localhost.32789      32768    0 32768    0
ESTABLISHED
...
...
Active UNIX domain sockets
Address  Type          Vnode      Conn  Local Addr      Remote Addr
3000070d988 stream-ord 30000a67708 00000000 /var/tmp/aaa3daGZa
3000070db20 stream-ord 300000fe7b0 00000000 /tmp/.X11-unix/X0
3000070dc8 stream-ord 00000000 00000000
#
```

Show Interface Multicast Group Memberships

```
# netstat -g
Group Memberships
Interface Group          RefCnt
-----
lo0      224.0.0.1        1
hme0    224.0.0.1        1
#
```

Show State of All TCP/IP Interfaces

```
# netstat -i
Name    Mtu  Net/Dest        Address          Ipkts  Ierrs Opkts  Oerrs
Collis Queue
lo0     8232 loopback      localhost        20140   0     20140   0     0
0
hme0    1500 rafael         rafael          860070  0     809700  0     0
0
#
```

Where:

- **Ipkts** – The number of packets received since the machine was booted.
- **Ierrs** – The number of input errors (should be less than 1 percent of **Ipkts**). If the input error rate is high (over 0.25 percent), the host may be dropping packets.
- **Opkts** – The number of packets transmitted since the machine was booted.
- **Oerrs** – The number of output errors (should be less than 1 percent of **Opkts**).
- **Collis** – The number of collisions experienced while sending packets. This is interpreted as a percentage of **Opkts**. A normal value is less than 1 percent. A network-wide collision rate greater than 5 to 10 percent can indicate a problem.

A machine with active network traffic should show both **Ipkts** and **Opkts** continually increasing.

Show Detailed Kernel Memory Allocation Information

```
# netstat -k
kstat_types:
raw 0 name=value 1 interrupt 2 i/o 3 event_timer 4

segmap:
fault 494755 faulta 0 getmap 728460 get_use 154 get_reclaim 656037
get_reuse 65235
get_unused 0 get_nofree 0 rel_async 15821 rel_write 16486 rel_free
376
rel_abort 0 rel_dontneed 15819 release 711598 pagecreate 64294
...
...
lo0:
ipackets 20140 opackets 20140

hme0:
ipackets 873597 ierrors 0 opackets 831583 oerrors 0 collisions 0
defer 0 framing 0 crc 0 sqe 0 code_violations 0 len_errors 0
ifspeed 100 buff 0 oflo 0 uflo 0 missed 0 tx_late_collisions 0
retry_error 0 first_collisions 0 nocarrier 0 inits 8 nocanput 0
allocbfail 0 runt 0 jabber 0 babble 0 tmd_error 0 tx_late_error 0
rx_late_error 0 slv_parity_error 0 tx_parity_error 0 rx_parity_error
0
slv_error_ack 0 tx_error_ack 0 rx_error_ack 0 tx_tag_error 0
rx_tag_error 0 eop_error 0 no_tmdu 0 no_tbufs 0 no_rbufs 0
rx_late_collisions 0 rbytes 543130524 obytes 637185145 multircv 34120
multixmt 0
brdcstrcv 47166 brdcstxmt 100 norcvbuf 0 noxmtbuf 0 phy_failures 0
...
...
lm_config:
buf_size 80 align 8 chunk_size 80 slab_size 8192 alloc 1 alloc_fail 0
free 0 depot_alloc 0 depot_free 0 depot_contention 0 global_alloc 1
global_free 0 buf_constructed 0 buf_avail 100 buf_inuse 1
buf_total 101 buf_max 101 slab_create 1 slab_destroy 0 memory_class 0
hash_size 0 hash_lookup_depth 0 hash_rescale 0 full_magazines 0
empty_magazines 0 magazine_size 7 alloc_from_cpu0 0 free_to_cpu0 0
buf_avail_cpu0 0

#
```

Show STREAMS Statistics

```
# netstat -m
streams allocation:
                                         cumulative
allocation
                                         current   maximum   total
failures
streams          287        312      15176
0
queues          722        765      34903
0
mblk            532       1651     691067
0
dblk            518       2693     6814776
0
linkblk          8         169       18
0
strevent         9         169     215517
0
syncq            14         67       77
0
qband            0         0         0
0

659 Kbytes allocated for streams data
#
```

Show Multicast Routing Tables

```
# netstat -M

Virtual Interface Table is empty

Multicast Forwarding Cache
Origin-Subnet           Mcastgroup    # Pkts  In-Vif  Out-
vifs/Forw-ttl

Total no. of entries in cache: 0
```

Show Multicast Routing Statistics

```
# netstat -Ms

multicast routing:
    0 hits - kernel forwarding cache hits
    0 misses - kernel forwarding cache misses
    0 packets potentially forwarded
    0 packets actually sent out
    0 upcalls - upcalls made to mrouted
    0 packets not sent out due to lack of resources
    0 datagrams with malformed tunnel options
    0 datagrams with no room for tunnel options
    0 datagrams arrived on wrong interface
    0 datagrams dropped due to upcall Q overflow
    0 datagrams cleaned up by the cache
    0 datagrams dropped selectively by ratelimiter
    0 datagrams dropped - bucket Q overflow
    0 datagrams dropped - larger than bkt size
```

Show Network Addresses as Numbers, Not Names

```
# netstat -n

TCP
  Local Address          Remote Address      Swind Send-Q Rwind Recv-Q
State
-----
-----
129.147.11.213.1023  129.147.4.111.2049    8760      0 8760      0
ESTABLISHED
127.0.0.1.32805       127.0.0.1.32789      32768      0 32768      0
ESTABLISHED
127.0.0.1.32789       127.0.0.1.32805      32768      0 32768      0
ESTABLISHED
...
...
129.147.11.213.34574  129.147.4.36.45430    8760      0 8760      0
CLOSE_WAIT
129.147.11.213.974    129.147.4.43.2049    8760      0 8760      0
ESTABLISHED
Active UNIX domain sockets
Address   Type      Vnode      Conn  Local Addr          Remote Addr
3000070d988 stream-ord 30000a67708 00000000 /var/tmp/aaa3daGZa
3000070db20 stream-ord 300000fe7b0 00000000 /tmp/.X11-unix/X0
3000070dc8 stream-ord 00000000 00000000
#
```

Show the Address Resolution Protocol (ARP) Tables

```
# netstat -p
Net to Media Table
Device    IP Address          Mask      Flags   Phys Addr
-----  -----
--- 
hme0     mandan-qfe1        255.255.255.255
08:00:20:93:9e:2d
hme0     129.147.11.248      255.255.255.255
00:00:0c:07:ac:00
hme0     micmac-qfe1        255.255.255.255
08:00:20:92:9f:c0
hme0     ssdlegal            255.255.255.255
08:00:20:8f:6d:eb
hme0     rafael              255.255.255.255 SP
08:00:20:78:54:90
hme0     224.0.0.0           240.0.0.0       SM
01:00:5e:00:00:00
#
```

Show the Routing Tables

```
# netstat -r
Routing Table:
Destination      Gateway          Flags  Ref  Use
Interface
-----  -----
----- 
129.147.11.0    rafael          U      3   165
hme0
224.0.0.0        rafael          U      3   0
hme0
default          129.147.11.248  UG     0   6097
localhost        localhost        UH     0   9591
lo0
#
```

The first column shows the destination network, the second the router through which packets are forwarded. The **U** flag indicates that the route is up; the **G** flag indicates that the route is to a gateway. The **H** flag indicates that the destination is a fully qualified host address, rather than a network. A **D** flag indicates the route was dynamically created using a ICMP-redirect.

The **Ref** column shows the number of active uses per route, and the **Use** column shows the number of packets sent per route. The **Interface** column shows the network interface that the route uses.

Show Per Protocol Statistics

```
# netstat -s

      UDP
      udpInDatagrams      = 30568    udpInErrors      =
0
      udpOutDatagrams      = 30537

      TCP
200      tcpRtoAlgorithm      = 4      tcpRtoMin      =
      tcpRtoMax      = 60000    tcpMaxConn      =
-1
      tcpActiveOpens      = 1177   tcpPassiveOpens      =
338      tcpAttemptFails      = 4      tcpEstabResets      =
5
      tcpCurrEstab      = 28     tcpOutSegs      =
=834720      tcpOutDataSegs      = 636268   tcpOutDataBytes      =
=612298133      tcpRetransSegs      = 57     tcpRetransBytes      =
53036      tcpOutAck      = 198438   tcpOutAckDelayed      =
26757
      ...
      ...
      IP
255      ipForwarding      = 2      ipDefaultTTL      =
      ipInReceives      = 809310   ipInHdrErrors      =
0
      ipInAddrErrors      = 0      ipInCksumErrs      =
0
      ipForwDatagrams      = 0      ipForwProhibits      =
0
      ipInUnknownProtos      = 0      ipInDiscards      =
0
      ipInDelivers      = 822773   ipOutRequests      =
=848209
      ...
      ...
      ICMP
0      icmpInMsgs      = 6      icmpInErrors      =
0
      icmpInCksumErrs      = 0      icmpInUnknowns      =
```

```

        icmpInDestUnreachs =      3      icmpInTimeExcds      =
0
...
...
IGMP:
    3237 messages received
        0 messages received with too few bytes
        0 messages received with bad checksum
    3237 membership queries received
        0 membership queries received with invalid field(s)
        0 membership reports received
        0 membership reports received with invalid field(s)
        0 membership reports received for groups to which we
belong
        0 membership reports sent
#

```

Show Extra Socket and Routing Table Information

```

# netstat -v

TCP
Local/Remote Address Swind   Snext      Suna     Rwind   Rnext      Rack   Rto     MSS
State
-----
-
rafael.1023
ra.nfsd          8760 b6e2967a b6e2967a 8760 09c399a6 09c399a6 405
1460 ESTABLISHED
localhost.32805
localhost.32789  32768 02adf2a5 02adf2a5 32768 02afdf90f 02afdf90f 4839
8192 ESTABLISHED
localhost.32789
localhost.32805  32768 02afdf90f 02afdf90f 32768 02adf2a5 02adf2a5 4818
8192 ESTABLISHED
localhost.32808
localhost.32802  32768 02b68d0b 02b68d0b 32768 02b71347 02b71347 433
8192 ESTABLISHED
localhost.32802
localhost.32808  32768 02b71347 02b71347 32768 02b68d0b 02b68d0b 406
8192 ESTABLISHED
localhost.32811
localhost.32810  32768 02b7d3ce 02b7d3ce 32768 02b86f76 02b86f76 462
8192 ESTABLISHED
localhost.32810
localhost.32811  32768 02b86f76 02b86f76 32768 02b7d3ce 02b7d3ce 3375
8192 ESTABLISHED
localhost.32814
localhost.32802  32768 02c1ae7e 02c1ae7e 32768 02c3aa4d 02c3aa4d 407
8192 ESTABLISHED
localhost.32802

```

```
localhost.32814      32768 02c3aa4d 02c3aa4d 32768 02c1ae7e 02c1ae7e  462
8192 ESTABLISHED
localhost.32817
localhost.32816      32768 02c37344 02c37344 32768 02c43cf7 02c43cf7  462
8192 ESTABLISHED
localhost.32816
...
...
localhost.32847      32768 02fb576a 02fb576a 32768 02fa9745 02fa9745  3375
8192 ESTABLISHED
rafael.32879
bast imap           8760 03935a31 03935a31  8760 3de1ed9d 3de1ed9d  452
1460 ESTABLISHED
rafael telnet
proto198.Central.Sun.COM.32831 8760 5f0321d6 5f0321d6 8760 029bella 029bella
457 1460 ESTABLISHED
rafael.34069
proto198.Central.Sun.COM.6000 8760 5f6577c1 5f6577c1 8760 02c75d3c 02c75d3c
457 1460 ESTABLISHED
rafael.34574
bast.45430          8760 85bc6ae2 85bc6ae2  8760 1496dc76 1496dc76  4778
1460 CLOSE_WAIT
rafael.973
shasta.nfsd         8760 affeda04 affeda04  8760 00c090b0 00c090b0  3719
1460 ESTABLISHED
Active UNIX domain sockets
Address  Type          Vnode    Conn Local Addr      Remote Addr
3000070d988 stream-ord 30000a67708 00000000 /var/tmp/aaa3daGZa
3000070db20 stream-ord 300000fe7b0 00000000 /tmp/.X11-unix/X0
3000070dcb8 stream-ord 00000000 00000000
#
```

nfsstat

Show Client Information Only

```
# nfsstat -c

Client rpc:
Connection oriented:
calls      badcalls    badxids    timeouts   newcreds   badverfs
149404        4          0          0          0          0
timers      cantconn    nomem     interrupts
0            4          0          0

Connectionless:
calls      badcalls    retrans    badxids    timeouts   newcreds
9           1          0          0          0          0
badverfs    timers     nomem     cantsend
0            4          0          0

Client nfs:
calls      badcalls    clgets     cltoomany
146831        1          146831    0

Version 2: (6 calls)
null       getattr    setattr    root       lookup     readlink
0 0%       5 83%      0 0%      0 0%      0 0%      0 0%
read       wrcache    write      create     remove     rename
0 0%       0 0%      0 0%      0 0%      0 0%      0 0%
link       symlink    mkdir      rmdir     readdir   statfs
0 0%       0 0%      0 0%      0 0%      0 0%      1 16%
Version 3: (144706 calls)
null       getattr    setattr    lookup     access     readlink
0 0%       42835 29%  5460 3%  25355 17%  18010 12%  742 0%
read       write      create     mkdir     symlink   mknod
18888 13%  23273 16%  2985 2%  188 0%   3 0%      0 0%
remove     rmdir      rename    link      readdir
readdirplus
1870 1%    5 0%      944 0%    17 0%    990 0%    744 0%
fsstat     fsinfo     pathconf  commit
73 0%      32 0%      275 0%    2017 1% 

Client nfs_acl:
Version 2: (1 calls)
null       getacl     setacl     getattr    access
0 0%       0 0%      0 0%      1 100%   0 0%

Version 3: (2118 calls)
null       getacl     setacl
0 0%       2118 100%  0 0%
#
```

Where:

- `calls` – The total number of calls sent.
- `badcalls` – The total number of calls rejected by RPC.
- `retrans` – The total number of retransmissions.
- `badxid` – The number of times that a duplicate acknowledgment was received for a single NFS request. If it is approximately equal to `timeout` and above 5 percent, look for a server bottleneck.
- `timeout` – The number of calls that timed out waiting for a reply from the server. If the value is more than 5 percent, then RPC requests are timing out. A `badxid` value of less than 5 percent indicates that the network is dropping parts of the requests or replies. Check that intervening networks and routers are working properly. Consider reducing the NFS buffer size parameters. (See the `mount_nfs` command and `rsize` and `wsize` options.) However, reducing these parameters will reduce peak throughput.
- `wait` – The number of times a call had to wait because no client handle was available.
- `newcred` – The number of times the authentication information had to be refreshed.
- `timers` – The number of times the time-out value was greater than or equal to the specified time-out value for a call.
- `readlink` – The number of times a read was made to a symbolic link. If this number is high (over 10 percent), it could mean that there are too many symbolic links.

Show NFS Mount Options

```
# nfsstat -m

/home/craigm from ra:/export/home13/craigm
Flags:
vers=3,proto=tcp,sec=sys,hard,intr,link,symlink,acl,rsize=32768,wsize=32768,r
etrans=5

/usr/dist from shasta,seminole,otero,wolfcreek:/usr/dist
Flags:
vers=3,proto=tcp,sec=sys,hard,intr,llock,link,symlink,acl,rsize=32768,wsize=3
2768,retrans=5
Failover:noreponse=0, failover=0, remap=0, currserver=shasta

/home/downhill from ra:/export/home19/downhill
Flags:
vers=3,proto=tcp,sec=sys,hard,intr,link,symlink,acl,rsize=32768,wsize=32768,r
etrans=5
#
```

Output includes the server name and address, mount flags, current read and write sizes, the retransmission count, and the timers used for dynamic retransmission.

Show NFS Information; Both Client and Server

```
# nfsstat -n

Server nfs:
calls      badcalls
0          0
Version 2: (0 calls)
null      getattr  setattr  root      lookup   readlink
0 0%      0 0%    0 0%    0 0%    0 0%    0 0%
read      wrcache  write    create   remove   rename
0 0%      0 0%    0 0%    0 0%    0 0%    0 0%
link      symlink mkdir    rmdir   readdir  statfs
0 0%      0 0%    0 0%    0 0%    0 0%    0 0%
Version 3: (0 calls)
null      getattr  setattr  lookup   access   readlink
0 0%      0 0%    0 0%    0 0%    0 0%    0 0%
read      write   create   mkdir   symlink  mknod
0 0%      0 0%    0 0%    0 0%    0 0%    0 0%
remove   rmdir   rename  link    readdir
readdirplus
0 0%      0 0%    0 0%    0 0%    0 0%    0 0%
fsstat   fsinfo   pathconf commit
0 0%      0 0%    0 0%    0 0%    0 0%    0 0%

Client nfs:
calls      badcalls  clgets   cltoomany
159727    1        159727  0
Version 2: (6 calls)
null      getattr  setattr  root      lookup   readlink
0 0%      5 83%   0 0%    0 0%    0 0%    0 0%
read      wrcache  write   create   remove   rename
0 0%      0 0%    0 0%    0 0%    0 0%    0 0%
link      symlink mkdir   rmdir   readdir  statfs
0 0%      0 0%    0 0%    0 0%    0 0%    1 16%
Version 3: (157404 calls)
null      getattr  setattr  lookup   access   readlink
0 0%      45862 29% 5622 3% 27178 17% 19391 12% 784 0%
read      write   create   mkdir   symlink  mknod
21439 13% 26273 16% 3224 2% 190 0% 4 0% 0 0%
remove   rmdir   rename  link    readdir
readdirplus
2008 1% 5 0%    1073 0% 17 0% 1012 0% 771 0%
fsstat   fsinfo   pathconf commit
73 0%    35 0%   288 0% 2155 1%
#
```

Show RPC Information

```
# nfsstat -r

Server rpc:
Connection oriented:
calls      badcalls      nullrecv      badlen      xdrcall
dupchecks
0          0            0            0            0            0
dupreqs
0
Connectionless:
calls      badcalls      nullrecv      badlen      xdrcall
dupchecks
2          0            0            0            0            0
dupreqs
0

Client rpc:
Connection oriented:
calls      badcalls      badxids      timeouts      newcreds      badverfs
162588    4            0            0            0            0
timers      cantconn      nomem        interrupts
0          4            0            0
Connectionless:
calls      badcalls      retrans      badxids      timeouts      newcreds
9          1            0            0            0            0
badverfs    timers      nomem        cantsend
0          4            0            0
#
```

Show Server Information only

```
# nfsstat -s

Server rpc:
Connection oriented:
calls      badcalls      nullrecv      badlen      xdrcall
dupchecks
0          0            0            0            0            0
dupreqs
0
Connectionless:
calls      badcalls      nullrecv      badlen      xdrcall
dupchecks
2          0            0            0            0            0
dupreqs
0

Server nfs:
calls      badcalls
0          0
Version 2: (0 calls)
null      getattr      setattr      root        lookup      readlink
0 0%      0 0%        0 0%        0 0%       0 0%        0 0%
read      wrcache      write        create      remove      rename
0 0%      0 0%        0 0%        0 0%       0 0%        0 0%
link      symlink      mkdir        rmdir      readdir      statfs
0 0%      0 0%        0 0%        0 0%       0 0%        0 0%
Version 3: (0 calls)
null      getattr      setattr      lookup      access      readlink
0 0%      0 0%        0 0%        0 0%       0 0%        0 0%
read      write        create      mkdir      symlink      mknod
0 0%      0 0%        0 0%        0 0%       0 0%        0 0%
remove    rmdir        rename      link       readdir
readdirplus
0 0%      0 0%        0 0%        0 0%       0 0%        0 0%
fsstat    fsinfo       pathconf    commit
0 0%      0 0%        0 0%        0 0%

Server nfs_acl:
Version 2: (0 calls)
null      getacl       setacl       setattr      access
0 0%      0 0%        0 0%        0 0%       0 0%
Version 3: (0 calls)
null      getacl       setacl
0 0%      0 0%        0 0%
#
```

Where:

- **calls** – The total number of RPCs received. NFS is just one RPC application.
- **badcalls** – The total number of RPC calls rejected. If this value is nonzero, then RPC requests are being rejected. Reasons include having a user in too many groups, attempts to access an unexported file system, or an improper secure RPC configuration.
- **nullrecv** – The number of times an RPC call was not there when one was thought to be received.
- **badlen** – The number of calls with length shorter than the RPC minimum.
- **xdrcall** – The number of RPC calls whose header could not be decoded by the external data representation (XDR) translation.
- **readlink** – If this value is more than 10 percent of the mix, then client machines are making excessive use of symbolic links on NFS-exported file systems. Replace the line with a directory, perhaps using a loopback mount on both server and clients.
- **getattr** – If this value is more than 60 percent of the mix, then check that the attribute cache value on the NFS clients is set correctly. It may have been reduced or set to zero. See the `mount_nfs` command and the `actimo` option.
- **null** – If this value is more than 1 percent, then the automounter time-out values are set too short. Null calls are made by the automounter to locate a server for the file system.
- **writes** – If this value is more than 5 percent, then configure NVRAM in the disk subsystem or a logging file system on the server.

Index

Symbols

/tmp 6-41
/usr/proc/bin 2-17

A

access times
 cache 5-27
accounting 2-23
 charging mechanisms C-4
connection C-3
customizing C-24
daily command
 summary C-15
daily report C-13
daily usage report C-14
definition C-2
disk C-4
enabling 11-4
files C-5
 disktacct C-10, C-27
 fee C-10
 fiscrptMM C-20
 pacct C-10, C-26
 rpptMMDD C-12
 summary C-28
 wtmp C-10, C-25
monthly total command
 summary C-17
periodic reports C-20
process C-3
report files C-7
starting and stopping C-8

summary C-28
uses C-2

acctcom
 pacct C-21

activity
 CPU 4-28

adb 2-24

address space 6-4

addressing
 SCSI 8-11

allocation
 data block 9-25
 file system
 performance 9-24

application
 I/O 9-30
 I/O statistics 9-65
 tuning 11-5
 tuning tips 11-7

application threads 3-12
 execution 3-14

atime 9-17

autoup 9-37

B

background execution 3-4

bandwidth
 definition 1-10
 network 10-4

block
 file system 9-5

board types 7-13

bottlenecks
 CPU 11-17
 I/O 11-13
 memory 11-10
 performance 11-8

BSD
 file system 9-5

bufhwm 9-19

bus
 characteristics 7-4
 definition 7-3
 Fibre Channel 8-15
 Gigaplane 7-7
 Gigaplane XB 7-8
 limits 7-21
 MBus 7-6
 overload 7-26
 PCI 7-12
 peripheral 7-11
 problems 7-23
 SBus 7-12
 SCSI 8-3
 tuning reports 7-29
 UPA 7-7
 XDbus 7-6

C

cable
 SCSI lengths 8-9

cache
 access times 5-27
 cachefs D-1
 characteristics 5-14
 definition 5-3
 direct mapped 5-17
 directory name 9-10
 disk drive 8-27
 file system metadata 9-19
 hardware 5-4
 Harvard 5-18
 hierarchy 5-25
 hit rate 5-11
 inode 9-12
 miss 5-8
 NFS D-1

operation 5-7
PDC 6-8
performance 5-13
performance issues 5-28
physical address 5-16
replacement 5-9
segmap 9-30
 using 9-33

set associative 5-17

snooping 5-21

thrashing 5-23

tuning 5-30

UFS size 9-14

unified 5-18

virtual address 5-15, 6-8

write-back 5-19

write-through 5-19

cachefs D-1
 benefits D-3
 CD-ROM D-11
 characteristics D-2
 exercise D-12
 limitations D-4
 mounting D-7
 operation D-8
 setting up D-5
 writes D-8

cachefslog D-9

cachefspack D-10

cachefsstat D-9

cachefswssize D-10

caches
 CPU 5-6

cancellation
 write 5-19

capabilities
 SyMON B-3

CD-ROM
 cachefs D-11

cfsadmin D-5

characteristics
 cache 5-14
 file system 9-5
 SBus A-3
 SCSI bus 8-5

chargefee C-6

ckpacct C-6
class
 interactive scheduling 4-14
 real-time scheduling 4-15
 scheduling 4-4
clock
 synchronization 3-24
 tick 3-24
clock algorithm 6-16
clock routine 3-23
commands
 /proc 2-17
 adb 2-24
 cachefslog D-9
 cachefspack D-10
 cachefsstat D-9
 cachefswssize D-10
 cfsadmin D-5
 chargefee C-6
 ckpacct C-6
 CPU monitoring 4-35
 dispadmin 4-12
 dodisk C-6
 fsck
 cachefs D-8
 fstyp 9-7, 9-63
 iostat 2-9
 ipcs E-3
 lastlogin C-6
 memstat 2-8
 modload E-3
 modunload E-3
 monacct C-6
 mpstat 2-10, 4-36
 ndd 2-24, 10-14
 netstat 2-11
 newfs 9-6, 9-63
 nfsstat 2-12
 nulladm C-6
 pbind 4-23
 prdaily C-6
 priocntl 4-19
 process manager 2-18
 prtconf 8-32
 psrset 4-23
 ptree 2-17
runacct C-6, C-17
scsiinfo 8-34
SyMON 2-13
sysdef 2-24
tunefs 9-63
vmstat 2-8
compiler optimization 11-6
components
 SunOS 1-16
connection accounting C-3
course
 objectives xxvi
 overview xx, xxi, xxii
course philosophy xxxviii
CPU
 activity 4-28
 bottlenecks 11-17
 caches 5-6
 mpstat reports 2-10
 run queue 4-27
 time reporting 4-28
CPU monitoring
 commands 4-35
CPU performance statistics 4-30
customizing accounting C-24
cylinder group 9-7

D

data block allocation 9-25
desfree 6-14
device properties
 SCSI 8-32
devices
 SBus capabilities A-4
direct I/O 9-39
direct mapped cache 5-17
directory
 allocation 9-23
 file system 9-9
 name cache 9-10
disk
 data transfer timing 8-23
 drive caching 8-27
 fast write 8-27
 features 8-24

-
- I/O performance
 - planning 8-36, 9-43
 - I/O time calculation 8-22
 - I/O time components 8-18
 - iostat service time 8-59
 - latency timing 8-23
 - multiple zone recording 8-25
 - ordered seeks 8-30
 - swap space 6-39
 - tagged queueing 8-29
 - tuning statistics 8-55
 - disk accounting C-4
 - dispadmin 4-12
 - dispatch
 - kernel threads 4-21
 - dispatch parameter table
 - interactive/timesharing 4-7
 - real-time 4-18
 - dispatch priorities 4-6
 - DLAT 6-8
 - DNLC 9-10
 - cache hit rate 9-34
 - size 9-14
 - dodisk C-6
 - DRAM 5-5
 - DVMA 7-14
 - dynamic reconfiguration 7-8,
7-28
 - processor sets 4-24
- E**
- elevator seeks 8-30
 - enable_grp_ism E-6
 - Ethernet
 - full duplex 10-5
 - exec 3-3
 - exit 3-3
- F**
- fast write
 - disk 8-27
 - fastscan 6-14
 - fencing
 - processors 4-23
 - Fibre Channel 8-15
- file system
 - allocation 9-23
 - performance 9-24
 - block size 9-5
 - directory 9-9
 - fragments 9-28
 - journaling 9-29
 - layout 9-6
 - local 9-5
 - logging 9-29
 - statistics 9-34
 - types 9-3
 - files
 - accounting C-5
 - fork 3-3
 - fragments 9-28
 - optimization 9-63
 - free memory queue 6-11
 - fsck
 - cachefs D-8
 - fsflush 9-37
 - fstyp 9-7, 9-63
 - full duplex Ethernet 10-5
- G**
- Gigaplane bus 7-7
 - Gigaplane XB 7-8
 - graphs
 - performance 1-12
 - guidelines 11-3
- H**
- handspreadpages 6-16
 - hard link 9-21
 - hardware caches 5-4
 - Harvard cache 5-18
 - hierarchy
 - cache 5-25
 - hit rate
 - cache 5-11
- I**
- I/O
 - application 9-30

application statistics 9-65
board types 7-13
bottlenecks 11-13
direct 9-39
disk time calculation 8-22
disk time components 8-18
DVMA 7-14
file system statistics 9-34
performance planning 8-36,
 9-43
priority paging 6-25
programmed 7-14
RAID 8-41
statistics 8-55
storage arrays 8-39
tape drives 8-38
tuning 8-52

idle
 CPU time 4-28

inode
 allocation 9-23
 block pointers 9-18
 cache 9-12
 size 9-14
 contents 9-16
 table 9-8
 time stamps 9-17

installing SyMON B-6

interactive scheduling class 4-14

interface card characteristics A-4

interleaving
 memory 7-18

interrupt
 levels 3-22

intimate shared memory E-6

ioctl 2-4

iostat 2-9
 service time 8-59

IP
 trunking 10-7

IPC
 ipcs E-3
 message queues E-11
 parameters
 changing E-4
 determining E-3

limits E-5
semaphores E-8
shared memory E-6
tunables E-1

ipcs E-3
ISM E-6

J

JBOD 8-39

journaling file system 9-29

K

kernel threads 3-13, 3-14
 dispatching 4-21

kstat 2-4

L

lastlogin C-6
latency
 disk timing 8-23

libraries
 shared 6-47

link
 hard 9-21
 soft 9-22
 symbolic 9-22

locklint 3-20

locks 3-18
 problems 3-20
 types 3-19

lockstat 3-21, 4-36

logging file system 9-29

lotsfree 6-14

LWP 3-14
 not swappable 6-36
 swapping 6-34
 swapping in 6-38

M

madvice 9-42

max_nprocs 3-8

maxbpg 9-64

maxpio 6-18

-
- maxuprc 3-9
 - maxusers 9-14
 - MBus 7-6
 - memory
 - bottlenecks 11-10
 - free queue 6-11
 - interleaving 7-18
 - SRAM and DRAM 5-5
 - tuning summary 6-63
 - virtual 6-3
 - memstat 2-8
 - message queues
 - parameters E-11
 - metadata cache 9-19
 - minfree 6-14, 9-64
 - miss
 - cache 5-8
 - mmap 9-41
 - mode pages 8-31
 - modload E-3
 - modunload E-3
 - monacct C-6
 - mount
 - atime 9-17
 - cachefs D-7
 - mpstat 2-10, 4-36
 - multiple zone recording 8-25
 - multithreading 3-10
- N**
- ncsize 3-9, 9-10
 - ndd 2-24, 10-5, 10-14
 - ndquot 3-9
 - netstat 2-11, 10-28
 - network
 - bandwidth 10-4
 - hardware performance 10-16
 - tuning 10-3
 - tuning reports 10-28
 - Network Time Protocol 3-24
 - newfs 9-6, 9-63
 - NFS 10-18
 - problems 10-27
 - server daemon threads 10-21
 - nfsstat 2-12, 10-28
- npty 3-9
 - NTP 3-24
 - nulladm C-6
- O**
- objectives
 - course xxvi
 - optimization
 - compiler 11-6
 - overload
 - SBus 7-27
 - overloaded bus 7-26
 - overview
 - course xx, xxi, xxii
 - SCSI bus 8-3
- P**
- pacct
 - acctcom C-21
 - packet switched bus 7-4
 - page daemon 6-12
 - processing 6-14
 - page descriptor cache 6-8
 - page scanner 6-14
 - paging 6-11
 - clock algorithm 6-16
 - default parameters 6-15
 - I/O 6-18
 - priority 6-25
 - review 6-48
 - statistics 6-27
 - parameters
 - autoup 9-37
 - bufhwm 9-19
 - default paging 6-15
 - desfree 6-14
 - dispatch parameter table
 - interactive/timesharing 4-7
 - real-time 4-18
 - enable_grp_ism E-6
 - fastscan 6-14
 - file system 9-63
 - handspreadpages 6-16
 - IPC E-1

- changing E-4
- limits E-5
- lotsfree 6-14
- max_nprocs 3-8
- maxbpg 9-64
- maxpio 6-18
- maxuprc 3-9
- message queues E-11
- minfree 6-14, 9-64
- ncsize 3-9, 9-10
- ndquot 3-9
- npty 3-9
- process 3-8
- pt_cnt 3-9
- scsi_options 8-7
- semaphores E-8
- setting 2-25
- shared memory E-6
- slowscan 6-14
- TCP tuning 10-12
- tune_t_fsflushr 9-37
- ufs_HW 9-57
- ufs_LW 9-57
- ufs_ninode 3-9, 9-12
- use_ism E-6
- viewing 2-24
- partition 9-5
 - layout 9-6
- pbind 4-23
- PCI bus 7-12
- PDC 6-8
- performance
 - application I/O 9-30
 - bottlenecks 11-8
 - cache 5-13, 5-28
 - conceptual model 1-6
 - CPU 4-30
 - data collection 2-3
 - file system
 - statistics 9-34
 - file system allocation 9-24
 - graphs 1-12
 - I/O planning 8-36, 9-43
 - network hardware 10-16
 - process 3-17
 - TCP 10-11
- terminology 1-10
- threads 3-16
- peripheral buses 7-11
- philosophy xxxviii
- physical address cache 5-16
- PIO 7-14
- prdaily C-6
- priocntl 4-19
- priority
 - dispatch 4-6
 - level 3-22
 - SCSI bus 8-14
 - swapping 6-37
- priority paging 6-25
- problems
 - NFS 10-27
- proc commands 2-17
- process
 - address space 6-4
 - execution 3-13
 - limit 3-5
 - manager 2-18
 - performance 3-17
 - tuneables 3-8
 - zombie 3-5
- process accounting C-3
- processor sets 4-23
 - dynamic reconfiguration 4-24
 - using 4-25
- proctool 2-18
- programmed I/O 7-14
- prtconf 7-16, 8-32
- prtdiag 7-16
- pt_cnt 3-9
- ptree 2-17

Q

- quadratic rehash 9-26

R

- RAID
 - levels 8-41
- read system call 9-31
- real-time scheduling
 - issues 4-16

- real-time scheduling class 4-15
- replacement
 - cache 5-9
- resolution
 - clock 3-23
- response time
 - definition 1-11
- run queue 4-27
- runacct C-6, C-17
 - states C-18
- S**
 - sample tuning task 1-13
 - SBus 7-12
 - overload 7-27
 - SBus capabilities A-3
 - scheduling
 - classes 4-4
 - interactive class 4-14
 - real-time class 4-15
 - real-time issues 4-16
 - states 4-3
 - SCSI
 - storage arrays 8-39
 - SCSI bus
 - addressing 8-11
 - characteristics 8-5
 - device properties 8-32
 - lengths 8-9
 - options 8-7
 - overview 8-3
 - properties summary 8-10
 - speeds 8-6
 - tape drives 8-38
 - target priorities 8-14
 - terminators 8-9
 - tuning statistics 8-55
 - widths 8-8
 - SCSI device
 - mode pages 8-31
 - scsi_options 8-7
 - scsiinfo 8-34
 - SE Toolkit 2-20
 - segmap cache 9-30
 - bypassing 9-39
 - using 9-33
- segment
 - types 6-6
- segments 6-4
- semaphores
 - parameters E-8
- server
 - SyMON supported B-4
 - server I/O boards 7-13
 - server threads
 - NFS 10-21
 - servers
 - SBus capabilities A-3
 - service time
 - definition 1-11
 - iostat 8-59
 - set associative cache 5-17
 - shared libraries 6-47
 - shared memory
 - parameters E-6
 - sircuit switched bus 7-4
 - slice 9-5
 - layout 9-6
 - slowscan 6-14
 - SNMP
 - SyMON configuration B-10
 - snooping
 - cache 5-21
 - soft link 9-22
 - Solaris Desktop Extensions 2-18
 - source code
 - tuning 11-5
 - SRAM 5-5
 - statistics
 - application I/O 9-65
 - CPU 4-30
 - file system 9-34
 - I/O 8-55
 - paging 6-27
 - swapping 6-43
 - sticky bit 9-17
 - storage array
 - architecture 8-40
 - storage arrays 8-39
 - summary
 - accounting C-28

SunOS
 components 1-16
superblock 9-7
swap space 6-39
swapping 6-34
 conditions 6-35
 in 6-38
 priorities 6-37
 statistics 6-43
symbolic link 9-22
SyMON 2-13
 capabilities B-3
 configuration issues B-13
 configuring B-8
 installing B-6
 SNMP configuration B-10
 stopping or removing B-12
 supported servers B-4
 upgrading B-5
synchronization
 clock 3-24
sysdef 2-24
system
 CPU time 4-28
system calls
 exec 3-3
 exit 3-3
 fork 3-3
 kstat 2-4
 mmap 9-41
 read 9-31
 wait 3-3
 write 9-31
system threads 3-15
 NFS 10-21

T

tagged queueing 8-29
tape drives 8-38
targets
 SCSI 8-11
TCP
 stack issues 10-11
 tuning parameters 10-12
TCP connections 10-9

terminator
 SCSI 8-9
terminology 1-10
thrashing
 cache 5-23
threads 3-10
 application 3-12
 dispatching kernel
 threads 4-21
 execution 3-14
 kernel 3-13, 3-14
 NFS server daemon 10-21
 performance 3-16
 synchronization 3-11
 system 3-15
throttle
 UFS write 9-57
throughput
 definition 1-11
time
 CPU 4-28
time slicing 4-5
time stamps
 inode 9-17
TLB 6-8
tmpfs 6-41
transfer
 disk timing 8-23
translation
 virtual address 6-7
trunking 10-7
truss 3-21
tune_t_fsflushr 9-37
tunable
 process 3-8
tunefs 9-63
tuning
 actions 11-4
 application tips 11-7
 basic procedure 1-5
 cache 5-30
 definition 1-4
 file system post-creation 9-63
 I/O 8-52
 memory 6-63
 network 10-3

- NFS 10-18
- sample task 1-13
- source code 11-5
- TCP 10-9
- terminology 1-10
- tips 11-21
- tuning guidelines 11-3
- tuning reports
 - bus 7-29
 - mpstat 4-36
 - netstat 10-28
 - network 10-28
 - nfsstat 10-28
- tuning tools
 - accounting 2-23
 - iostat 2-9
 - memstat 2-8
 - mpstat 2-10
 - netstat 2-11
 - nfsstat 2-12
 - process manager 2-18
 - SE Toolkit 2-20
 - SyMON 2-13
 - vmstat 2-8

U

- UFS
 - write throttle 9-57
- ufs_HW 9-57
- ufs_LW 9-57
- ufs_ninode 3-9, 9-12
- unified cache 5-18
- UPA bus 7-7
- use_ism E-6
- user
 - CPU time 4-28
- utilization
 - definition 1-11

V

- virtual address
 - cache 5-15
 - caching 6-8
 - lookup 6-10
 - translation 6-7

- Virtual Adrian 2-20
- virtual memory 6-3
 - /tmp 6-41
 - madvise 9-42
 - mmap 9-41
 - segments 6-4
 - tuning summary 6-63
- vmstat 2-8
- vnode
 - interface 9-3
- vxstat 2-9

W

- wait 3-3
- wait I/O
 - CPU time 4-28
- write
 - cachefs D-8
 - cancellation 5-19
- write system call 9-31
- write throttle
 - UFS 9-57
- write-back cache 5-19
- write-through cache 5-19

X

- XDbus 7-6

Z

- zombie process 3-5

Copyright 1999 Sun Microsystems Inc., 901 San Antonio Road, Palo Alto, Californie 94303, U.S.A. Tous droits réservés.

Ce produit ou document est protégé par un copyright et distribué avec des licences qui en restreignent l'utilisation, la copie, la distribution, et la décompilation. Aucune partie de ce produit ou de sa documentation associée ne peut être reproduite sous aucune forme, par quelque moyen que ce soit, sans l'autorisation préalable et écrite de Sun et de ses bailleurs de licence, s'il y en a.

Des parties de ce produit pourront être dérivées du système UNIX® licencié par Novell, Inc. et du système Berkeley 4.3 BSD licencié par l'Université de Californie. UNIX est une marque enregistrée aux Etats-Unis et dans d'autres pays et licenciée exclusivement par X/Open Company Ltd. Le logiciel détenu par des tiers, et qui comprend la technologie relative aux polices de caractères, est protégé par un copyright et licencié par des fournisseurs de Sun.

Sun, Sun Microsystems, le logo Sun, Solaris, NFS, SunOS, Solstice SyMON, HotJava, OpenWindows, OpenWindows, OpenBoot, Solstice DiskSuite, Sun Trunking, SunVTS, Ultra Enterprise, Sun StorEdge, SunNet, CacheFS, SunSwift, SunVideo, et VideoPix sont des marques déposées ou enregistrées de Sun Microsystems, Inc. aux Etats-Unis et dans d'autres pays. Toutes les marques SPARC, utilisées sous licence, sont des marques déposées ou enregistrées de SPARC International, Inc. aux Etats-Unis et dans d'autres pays. Les produits portant les marques SPARC sont basés sur une architecture développée par Sun Microsystems, Inc.

Les interfaces d'utilisation graphique OPEN LOOK® et Sun™ ont été développées par Sun Microsystems, Inc. pour ses utilisateurs et licenciés. Sun reconnaît les efforts de pionniers de Xerox pour la recherche et le développement du concept des interfaces d'utilisation visuelle ou graphique pour l'industrie de l'informatique. Sun détient une licence non exclusive de Xerox sur l'interface d'utilisation graphique Xerox, cette licence couvrant aussi les licenciés de Sun qui mettent en place l'interface d'utilisation graphique OPEN LOOK et qui en outre se conforment aux licences écrites de Sun.

Le système X Window est un produit de X Consortium, Inc.

CETTE PUBLICATION EST FOURNIE "EN L'ETAT" SANS GARANTIE D'AUCUNE SORTE, NI EXPRESSE NI IMPLICITE, Y COMPRIS, ET SANS QUE CETTE LISTE NE SOIT LIMITATIVE, DES GARANTIES CONCERNANT LA VALEUR MARCHANDE, L'APTITUDE DES PRODUITS A RÉPONDRE A UNE UTILISATION PARTICULIERE, OU LE FAIT QU'ILS NE SOIENT PAS CONTREFAISANTS DE PRODUITS DE TIERS.



Please
Recycle



Adobe PostScript

