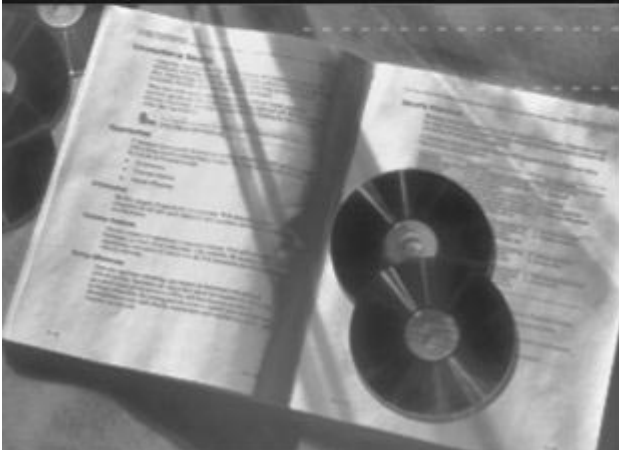


OpenVMS System Performance Management

Student Guide

Training



COMPAQ

OpenVMS System Performance Management

Student Guide

Training



COMPAQ

Notice

© 1999 Compaq Computer Corporation

COMPAQ, the Compaq logo, and the Digital logo are registered in the U.S. Patent and Trademark Office. Other product names mentioned herein may be the trademarks of their respective companies.

Confidential computer software. Valid license from Compaq or authorized sublicensor required for possession, use, or copying. Consistent with FAR 12.211 and 12.212, Commercial Computer Software, Computer Software Documentation, and Technical Data for Commercial Items are licensed to the U.S. Government under vendor's standard commercial license.

Compaq Computer Corporation shall not be liable for technical or editorial errors or omissions contained herein. The information in this document is subject to change without notice.

Aero, ALPHA, ALPHA AXP, AlphaServer, AlphaStation, Armada, BackPaq, COMPAQ, Compaq Insight Manager, CompaqCare logo, Counselor, DECterm, Deskpro, DIGITAL, DIGITAL logo, DIGITAL Alpha Systems, Digital Equipment Corporation, DIGITAL UNIX, DirectPlus, FASTART, Himalaya, InfoPaq, Integrity, LicensePaq, Ministation, NetFlex, NonStop, OpenVMS, PaqFax, Presario, ProLiant, ProLinea, ProSignia, QuickBack, QuickFind, Qvision, RDF, RemotePaq, RomPaq, ServerNet, SERVICenter, SmartQ, SmartStart, SmartStation, SolutionPaq, SpeedPaq, StorageWorks, Systempro/LT, Tandem, TechPaq, TruCluster, and Tru64 UNIX are registered in United States Patent and Trademark Office.

Atalla, C-Series, Expand, FOX, Guardian, iTP, Measure, Netelligent, and PointView are trademarks of Compaq Computer Corporation.

Microsoft, Windows, and Windows NT are registered trademarks of Microsoft Corporation.

Intel, Pentium, and Intel Inside are registered trademarks and Xeon is a trademark of Intel Corporation.

UNIX is a trademark in the US and other countries, licensed exclusively through X-Open Company Ltd.

OpenVMS System Performance Management

Student Guide

December 1999

Course Overview

Course Description	Overview-1
Course Length	Overview-1
Intended Audience	Overview-1
Course Content and Objectives	Overview-1
Course Outline	Overview-2
Course Goals	Overview-7
Course Nongoals	Overview-8

Module 1 — General Performance Topics

Module Outline	1-1
Objectives	1-1
Topics	1-1
Performance Issues	1-2
Performance Goals	1-2
Performance Tradeoffs	1-2
Performance Metrics	1-2
Performance Bottlenecks	1-3
System Resources	1-3
Tuning Strategies	1-3
OpenVMS Performance Metrics	1-4
Relative Units of Time	1-4
OpenVMS Performance Tools	1-5
Monitoring Tools	1-5
Management Tools	1-5

Module 2 — Layout of Virtual Address Space

Module Outline	2-1
Objectives	2-1
Topics	2-1
Layout of Virtual Address Space	2-2
OpenVMS Alpha Memory Layout	2-3
Address Space	2-4
OpenVMS Alpha Memory Management	2-5
Memory Terms	2-5
Address Space	2-5
P0 Space Contents	2-6
Image	2-6
Shareable Images	2-6
Debugger and Traceback Handler	2-6

P1 Space Layout	2-7
Stacks	2-7
Command Language Interpreter (CLI)	2-7
CLI Symbol Table	2-7
Process Allocation Region	2-7
Channel Control Block Table	2-7
RMS Buffers	2-7
System Space Layout (Alpha)	2-8
S2 Space (Start: FFFFFFFE.00000000)	2-8
S0/S1 Space (Start: FFFFFFFF.80000000)	2-8
System Space	2-9
PFN Database	2-9
Balance Slot Area	2-9
System Working Set List	2-10
System Page Table	2-10
Paged Pool	2-10
S0/S1 Space	2-11
Nonpaged Pool	2-11
Monitoring Pool	2-12
Nonpaged Pool Reclamation	2-13
Pool Zones	2-19
Lock Manager Use of Pool Zones	2-20

Module 3 — Image Activation and Paging

Module Outline	3-1
Objectives	3-1
Topics	3-1
Paging Compared to Swapping	3-2
Paging	3-2
Swapping	3-2
Address Translation	3-3
Address Translation Model	3-3
Page Faults	3-4
Locating Pages in Virtual Memory	3-5
Producing an Image File	3-6
Image Header	3-6
Image Body	3-6
Fixup Vector	3-6
Image Activation and Paging	3-7
Process Section Table Entries	3-7
Page Table Entries	3-7
Process Section Table Index	3-8
Demand Zero Entry	3-8
Global Page Table Index	3-9
Image Activation	3-10
Free Page List	3-11

Valid Pages and the Working Set List	3-12
Working Set List.....	3-12
Displacing a Page from the Working Set List.....	3-12
Free List Fault Rate	3-13
Modified List Fault Rate	3-14
Secondary Hard Faults.....	3-15
Process Section Table Index	3-15
Page File Virtual Block Number	3-15
Secondary Hard Faults.....	3-16

Module 4 — Interpreting Paging Metrics

Module Outline.....	4-1
Objectives.....	4-1
Topics.....	4-1
Sample Display from \$Monitor Page	4-2
Interpreting the \$Monitor Page Display.....	4-3
Page Fault Rate.....	4-3
Read I/O Rate.....	4-3
Page Read Rate.....	4-3
Page Write I/O Rate.....	4-3
Page Write Rate.....	4-3
Free List Fault Rate	4-3
Modified List Fault Rate.....	4-3
Demand Zero Fault Rate.....	4-4
Global Valid Fault Rate	4-4
Wrt-In-Progress Fault Rate	4-4
System Fault Rate.....	4-4
Free List Size.....	4-4
Modified List Size	4-4
Interpreting Paging Metrics.....	4-5
Paging Metrics Alpha vs. VAX.....	4-8
Alpha \$MONITOR Page	4-9

Module 5 — Working Sets and Automatic Working Set Adjustment

Module Outline.....	5-1
Objectives.....	5-1
Topics.....	5-1
Working Set List.....	5-2
Working Set Size vs. Working Set List Size.....	5-3
Working Set	5-3
Working Set List.....	5-3
Parameters Affecting the Working Set List Size.....	5-4
Working Set Default or Limit	5-4
Working Set Quota	5-4
Working Set Extent	5-5
Automatic Working Set Adjustment	5-6
AWSA Mechanics	5-7

Working Set Performance Considerations	5-9
Setting WSDEFAULT	5-9
Setting WSQUOTA	5-9
AWSA Performance Considerations	5-10
Sizing WSEXTENT	5-10
BORROWLIM and GROWLIM	5-10
Evaluating Working Sets Example	5-11
Alpha AWSA Parameters	5-12
Sample Tuning Session	5-14
Workstation Parameter Restrictions	5-31

Module 6 — Locality and Paging Performance

Module Outline	6-1
Objectives	6-1
Topics	6-1
Locality	6-2
Locality Issues	6-3
Importance of Locality	6-4
Effects of Poor Locality	6-5
Effect of Improving Locality	6-8
Sample FORTRAN Programs	6-9
C Locality Programs	6-10

Module 7 — Shared Image Costs and Benefits

Module Outline	7-1
Objectives	7-1
Topics	7-1
Shared Image Costs and Benefits	7-2
Shared Image Costs	7-2
Global Section Descriptors	7-2
Global Section Table Entries	7-3
Global Page Table Entries	7-3
Global Paging Benefits	7-4
Image Activation and Paging	7-5
Page Fault Rate for Images Installed /SHARE	7-5
Shared Image Benefits	7-6
Shareable Image Characteristics	7-7
Linking Alpha Shareable Images	7-8
Alpha Shareable Images	7-9
Alpha Shareable Images Example	7-10
Alpha Resident Images	7-16
Section Binding	7-16
Alpha Resident Images Example	7-17
Alpha Sliced Executable	7-22
AXP Resident Executable Example	7-23

Shared Address Data.....	7–24
Shared Address Data Implementation	7–24
Limits to Shared Address Data.....	7–25
Shared Address Data Benefits Example	7–26
Shared Address Data Dependencies Example	7–29
Monitoring Image Region Pages.....	7–33
Shared and Other Install Options.....	7–34

Module 8 — Modified Page List and Modified Page Writing

Module Outline.....	8–1
Objectives.....	8–1
Topics.....	8–1
Modified Page List	8–2
Modified Page List Functions	8–2
Modified Page List Parameters	8–2
MPW_HILIMIT	8–2
MPW_LOLIMIT	8–2
Modified Page List Parameters and Metrics	8–3
Modified Page List Parameters	8–3
MPW_WRTCLUSTER	8–3
MPW_WAITLIMIT	8–3
MPW_THRESH.....	8–3
MPW_IOLIMIT	8–3
Modified Page Writing Metrics.....	8–4
Modified Page List Performance Considerations.....	8–5
Parameter Sizing.....	8–5
Modified List Faults and Working Set List Sizes	8–5
MPL Performance and Analysis.....	8–6
Multiple Page Files.....	8–6
Inefficient Modified Page Writing	8–6
Indicators of Modified List Problems.....	8–6
Page File Exhaustion	8–7
Modified Page Writer Parameter Relationships	8–8

Module 9 — The Swapper and Memory Reclamation

Module Outline.....	9–1
Objectives.....	9–1
Topics.....	9–1
The Swapper and Memory Reclamation.....	9–2
Activities for Memory Reclamation	9–3
Proactive Memory Reclamation Example	9–4
Memory Reclamation and Virtual I/O Cache	9–6
Interaction Between Virtual I/O and Proactive Memory Reclamation	9–6
Free Page List Parameter Relationships	9–7

Module 10 — System Faults

Module Outline.....	10-1
Objectives.....	10-1
Topics.....	10-1
System Faults	10-2
The Nature of System Faults.....	10-2
How Many System Faults Are Too Many?.....	10-2

Module 11 — General I/O Flow

Module Outline.....	11-1
Objectives.....	11-1
Topics.....	11-1
General I/O Flow	11-2
Geometry of Disks.....	11-4
Time-Based Components of a Disk Transfer	11-6
Optimization of Disk Operations.....	11-7
Virtual I/O Cache.....	11-8

Module 12 — Analyzing the Effects of Fragmentation

Module Outline.....	12-1
Objectives.....	12-1
Topics.....	12-1
Determining I/O Rates	12-2
Dump of Header for X.Dat	12-4
Effect of Making File Contiguous.....	12-10
Effect of Making Both Files Contiguous	12-13
Using Multiple Spindles.....	12-16

Module 13 — Controller Optimizations

Module Outline.....	13-1
Objectives.....	13-1
Topics.....	13-1
Controller Optimizations.....	13-2
Hierarchical Storage Controllers (HSx).....	13-2
Locating Bottlenecks	13-5
Other Optimizatons Through Layered Products	13-6
Volume Shadowing	13-6
Disk Striping	13-6
RAID.....	13-7

Module 14 — Files-11 ODS-2 Concepts

Module Outline.....	14-1
Objectives.....	14-1
Topics.....	14-1
Files-11 Terminology.....	14-2
Volumes	14-2
Files.....	14-2
Extents	14-2
Mapping Pointers.....	14-2
File Headers.....	14-2
Clusters	14-3
File IDs.....	14-3
Directory Concepts	14-4
Master File Directory (MFD).....	14-4
User File Directories (UFD).....	14-4
Sub File Directories (SFD)	14-4
File Open Operation.....	14-5

Module 15 — Contiguity and the File System

Module Outline.....	15-1
Objectives.....	15-1
Topics.....	15-1
Contiguity and the File System	15-2
Window Turns.....	15-2
Preventing Fragmentation	15-4
Contiguous-Best-Try	15-4
Contiguous	15-4
Extent Cache	15-4
Split Transfers	15-5
Window Turns Versus Split Transfers.....	15-6
Accessing Sequential Files.....	15-7
Accessing Files Randomly.....	15-8

Module 16 — Files System Caches

Module Outline.....	16-1
Objectives.....	16-1
Topics.....	16-1
File System Caches.....	16-2
File System Cache Parameters	16-2
Examining File System Caching Parameters	16-5
\$MONITOR File	16-6
Interpreting the \$MONITOR File.....	16-7
Shared and Private Caches	16-8
Checking for Shrunk Caches	16-9

Module 17 — RMS Structures and Design Considerations

Module Outline.....	17-1
Objectives.....	17-1
Topics.....	17-1
RMS File Concepts.....	17-2
Summary of RMS File Organizations	17-3
File Structures.....	17-4
Sequential File Organization.....	17-4
Relative Files.....	17-4
Indexed Sequential Files	17-4
Indexed File Organization for a File with No Alternate Keys	17-5
RMS Interfaces for Affecting File Structure.....	17-6
General File-Oriented Routines.....	17-6
Record Processing Routines.....	17-6
Structures Defining File/Record Characteristics to Routines	17-6
Bucket Splits.....	17-7
Record Reference Vectors (RRVs)	17-7
Index Bucket Splits.....	17-7
Indexed File Growth.....	17-8
Factors Affecting Bucket Splits.....	17-10
Population Techniques.....	17-10
Fill Factor.....	17-10
Bucket Size.....	17-10
Number of Record Additions Over Time	17-10
Duplicate Keys for Data Records	17-11
Compression.....	17-12
Data Key Compression	17-12
Data Record Compression	17-12
Index Compression	17-12

Module 18 — RMS Utilities

Module Outline.....	18-1
Objectives.....	18-1
Topics.....	18-1
File Definition Language	18-2
Creating an FDL File	18-2
Creating a Text File	18-2
Techniques for Creating Data Files Using an FDL File	18-3
Sample EDIT/FDL Session.....	18-4
Creating a Data File from an FDL File	18-7
Optimizing RMS Indexed Files.....	18-8
General Optimization Techniques.....	18-8
FDL Optimization.....	18-8

Examining File Structures.....	18–9
Example Using RMS Utilities to Tune an Indexed File Structure	18–10
Example Testing Random Access of Indexed Files	18–28

Module 19 — RMS I/O Buffering Considerations

Module Outline.....	19–1
Objectives.....	19–1
Topics.....	19–1
Buffers and Buckets.....	19–2
Sequential Files	19–2
Relative Files.....	19–2
Indexed Files	19–2
Design Considerations	19–3
Multiblock Count Example	19–4
Read Ahead/Write Behind	19–6
Read Ahead	19–6
Write Behind	19–6
Deferred Write.....	19–7
Setting Up Multiple Buffers.....	19–8
Determining the Number of Local Buffers	19–10
Sequential Files	19–10
Relative Files.....	19–10
Indexed Files	19–10
Local Buffer Example (Caching Disabled).....	19–11
Local Buffer Example (Caching Enabled).....	19–13
Global Buffers	19–16
Global Buffers Example	19–17

Module 20 — MONITOR RMS

Module Outline.....	20–1
Objectives.....	20–1
Topics.....	20–1
MONITOR RMS	20–2
\$MONITOR RMS	20–2
Locking	20–3
Caching	20–3
All	20–3
MONITOR RMS Example.....	20–4

Module 21 — Understanding CPU Performance

Module Outline.....	21–1
Objectives.....	21–1
Topics.....	21–1

CISC vs. RISC Architectures	21-2
Computer Architectures	21-2
CISC Architecture Goals	21-2
CISC Characteristics.....	21-2
RISC Architecture Goals	21-2
RISC Characteristics.....	21-3
Understanding CPU Performance	21-4
VAX Alignment Considerations Example.....	21-6
Data Alignment Issues.....	21-7
Nonaligned Data Example	21-9
Nonaligned and Compiler Aligned Data Sample Runs	21-10
Aligned Data Example.....	21-11
Badly Aligned Macro Example	21-12
Well Aligned Macro Example.....	21-14
C Code with Bad Alignment	21-16
C Small Data Accesses Issues.....	21-17
C Code with Good Alignment.....	21-18
Debugger Issues.....	21-19
Using the Break/Unaligned Debugger Option	21-20
CPU Performance and Symmetric Multiprocessing.....	21-21
Galaxy Performance Options	21-22
Galaxy Computing Model.....	21-23
Partitioned	21-23
Adaptive	21-23
Galaxy Shared Everything Model	21-24
CPU Migration Under Galaxy.....	21-25
Sample CPU Migration Using DCL	21-26
CPU Controls Using DCL.....	21-27

Module 22 — Understanding Time Spent in Modes

Module Outline.....	22-1
Objectives.....	22-1
Topics.....	22-1
OpenVMS Access Modes	22-2
Kernel Mode.....	22-4
Page Faults	22-4
Lock Requests	22-4
Logical Name Translations	22-5
I/O Activity	22-5
File System Requests.....	22-5
Executive Mode.....	22-6
Supervisor Mode	22-6
User Mode.....	22-7
Interrupt Stack/State Time	22-8
MP Synchronization Time	22-10

Idle Time	22–11
Compatibility Mode Time.....	22–12
Examining Time Spent in Access Modes	22–13
Analyzing Time Spent in Modes.....	22–14

Module 23 — OpenVMS Scheduling and Priorities

Module Outline.....	23–1
Objectives.....	23–1
Topics.....	23–1
Quantum and CPU Time	23–2
Priorities	23–3
Differences in Scheduling Between Priority Ranges.....	23–3
Scheduling States.....	23–4
MONITOR States.....	23–5
Computable Processes	23–6
Waiting States.....	23–7
Voluntary Wait States.....	23–8
Involuntary Wait States	23–8
States of Concern.....	23–9
MONITOR States.....	23–10

Module 24 — PIXSCAN and DORMANTWAIT

Module Outline.....	24–1
Objectives.....	24–1
Topics.....	24–1
PIXSCAN and DORMANTWAIT	24–2

Module 25 — Locking Concepts

Module Outline.....	25–1
Objectives.....	25–1
Topics.....	25–1
Locking Concepts	25–2
Other Characteristics of Locks	25–3
Application Design and Locking Performance	25–4
Locking Considerations	25–6
Record Options in RAB for Locking Shared Files.....	25–7
Sample \$MONITOR LOCK/Local Buffers	25–8
\$MONITOR LOCK Display (Global Buffers)	25–9
Interpreting the \$MONITOR LOCK Display	25–10
Lock Manager Parameters.....	25–11

Module 26 — Distributed Locking Concepts

Module Outline.....	26–1
Objectives.....	26–1
Topics.....	26–1
Distributed Locking Concepts.....	26–2

LOCKDIRWT SYSGEN Parameter.....	26–6
Meaning of LOCKDIRWT Before OpenVMS V4.6.....	26–6
OpenVMS V4.6 to V4.7 Meaning of LOCKDIRWT.....	26–6
OpenVMS V5.0 to V5.2 Meaning of LOCKDIRWT.....	26–7
OpenVMS V5.3 and Later Meaning of LOCKDIRWT	26–7
PE1.....	26–7
SDA Lock Statistics Example	26–8
Performance Issues with Distributed Locking	26–10
Distributed Locking Example	26–12

Module 27 — MSCP Server

Module Outline.....	27–1
Objectives.....	27–1
Topics.....	27–1
MSCP Server.....	27–2
Interpreting \$MONITOR MSCP	27–5

Appendices

Appendix A — Tailoring OpenVMS
Appendix B — Virtual Balance Slots
Appendix C — Fast I/O

Course Description

Course Length

4.5 days.

Intended Audience

OpenVMS system managers, applications programmers, and system programmers.

Course Content and Objectives

This course is intended to present OpenVMS performance under three subsystems: Memory Management, I/O, and CPU. The students will learn to isolate performance bottlenecks to one of the three subsystems. Students will learn the effects of SYSGEN parameters on each of the subsystems, when to change these parameters, and when other management modifications or the purchase of new hardware will be required. Students will learn sufficiency (keeping the system running), as well as tuning considerations. The relative merits/drawbacks of using AUTOGEN will also be described.

Course Outline

- General Performance Topics
 - Performance Issues
 - Performance Bottlenecks
 - OpenVMS Performance Tools
- Layout of Virtual Address Space
 - Layout of Virtual Address Space
 - P0 Space
 - P1 Space
 - S0/S1/S2 Space
- Image Activation and Paging
 - Page Faults
 - Producing an Image File
 - Image Activation and paging
 - Valid Pages and the Free and Modified Page Lists
 - The Free List Fault Rate
 - The Modified List Fault Rate
 - Secondary Hard Faults
- Interpreting Paging Metrics
 - Sample Display from \$MONITOR Page
 - Interpreting the \$MONITOR Page Display
 - Interpreting Paging Metrics
- Working Sets and Automatic Working Set Adjustment
 - The Working Set List
 - Working Set Size vs. Working Set List Size
 - Parameters Affecting the Working Set List Size
 - Working Set List Parameter Relationships
 - Automatic Working Set Adjustment
 - AWSA Mechanics
 - Working Set Performance Considerations
 - AWSA Performance Considerations

- Locality and Paging Performance
 - Locality
 - Locality Issues
 - Effects of Poor Locality
 - Effect of Improving Locality
- Shared Image Costs and Benefits
 - Shared Image Costs and Benefits
 - Shared and Other Install Benefits
 - Image Activation and Paging
 - Shared Image Benefits
 - Resident Images
 - Shared Address Data
- The Modified Page List and Modified Page Writing
 - The Modified Page List
 - Modified Page List Parameters and Metrics
 - Modified Page List Performance Considerations
 - MPL Performance and Analysis
- The Swapper and Memory Reclamation
 - The Swapper and Memory Reclamation
- System Faults
 - System Faults
- General I/O Flow
 - General I/O Flow
 - Geometry of Disks
 - Time-Based Components of a Disk Transfer
 - Optimization of Disk Transfers
 - Virtual I/O Cache
 - RMS File Concepts
- Analyzing the Effects of Fragmentation
 - Fragmentation
 - Spindle Contention

- Controller Optimizations
 - Controller Optimizations
 - Locating Bottlenecks
 - Other Optimizations Through Layered Products
 - Volume Shadowing
 - Disk Striping
 - RAID
- Files-11 ODS-2 Concepts
 - Files-11 Terminology
 - Directory Concepts
 - Files-11 Reserved Files
 - File Open Operation
- Contiguity and the File System
 - Window Turns
 - Split Transfers
- File System Caches
 - File System Caches
- RMS Structures and Design Considerations
 - Structure for Record Organizations
 - File Organizations
 - Sequential and Relative File Structure
 - Indexed File Organization
 - Indexed File Structure Descriptions
 - Structures Supporting Alternate Keys
 - RMS Interfaces for Affecting File Structure
 - Factors Affecting Bucket Splits
 - Duplicate Keys for Data Records
 - Compression
 - Alignment Considerations

- RMS Utilities
 - File Definition Language
 - Techniques for Creating Data Files Using an FDL File
 - Creating a Data File from an FDL File
 - Optimizing RMS Indexed Files
- RMS I/O Buffering Considerations
 - Buffers and Buckets
 - Read Ahead
 - Write Behind
 - Deferred Write
 - Setting Up Multiple Buffers
 - Determining the Number of Local Buffers
- MONITOR RMS
 - MONITOR RMS
- Understanding CPU Performance
 - Understanding CPU Ratings
 - CPU Performance and Symmetric Multiprocessing
- Understanding Time Spent in Modes
 - Kernel Mode
 - Executive Mode
 - Supervisor Mode
 - User Mode
 - Interrupt Stack/State Time
 - MP Synchronization time
 - Idle Time
 - Compatibility mode Time
 - Analyzing Time Spent in Modes
- OpenVMS Scheduling and Priorities
 - Quantum and CPU Time
 - Priorities
 - States of Concern

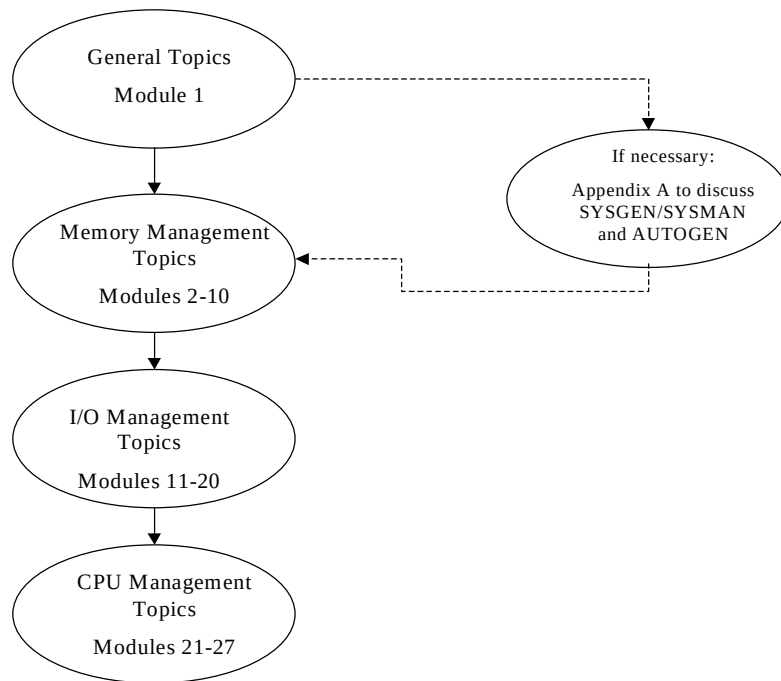
- PIXSCAN and DORMANTWAIT
 - PIXSCAN
 - DORMANTWAIT
- Locking Concepts
 - Locking Concepts
 - Other Characteristics of Locks
 - Application Design and Locking Performance
 - Interpreting the \$MONITOR LOCK Display
- Distributed Locking Concepts
 - Distributed Locking Concepts
 - The LOCKDIRWT SYSGEN Parameter
 - Performance Issues with Distributed Locking
- The MSCP Server
 - The MSCP Server
 - Interpreting \$MONITOR MSCP

Course Goals

- Students will be able to interpret performance metrics supplied by:
 - The MONITOR utility
 - DCL Commands including:
 - ◆ \$SHOW SYSTEM
 - ◆ \$SHOW STATUS
 - ◆ \$SHOW MEMORY
 - The INSTALL utility
- Students will be able to identify bottlenecks in the following subsystems:
 - Memory Management
 - I/O
 - CPU
- Students will be able to size working set parameters.
- Students will be able to monitor page file space.
- Students will be able to describe the advantages of balancing disk I/O.
- Students will be able to improve the performance characteristics of indexed files using the CONVERT utility.
- Students will be able to identify factors contributing to time spent executing in processor modes.
- Students will be able to describe file locking design considerations to attain good performance.
- Students will be able to monitor distributed lock traffic and impact its performance.
- Students will be able to identify the impact of changing key system parameters.

Course Nongoals

- Network performance
- Finding a magic system parameter that will make your system run faster.



General Performance Topics

Module 1

Module Outline

Objectives

This module discusses general topics in performance analysis that hold true regardless of the system you are running. The intent of the module is to get students on common ground as we move into the details of OpenVMS performance.

Topics

- Performance Issues
- Performance Bottlenecks
- OpenVMS Performance Tools

Performance Issues

Performance Goals

- Efficiency
- Sufficiency
- Effectiveness

Performance Tradeoffs

- Throughput
- Response time

Performance Metrics

- Arrival rate
- Service time

Performance Bottlenecks

Bottlenecks in a resource may cause degradation of the ability to use other resources within the system.

System Resources

Resource	Speed	Cost
CPU	Very fast	High
Memory	Fast	Moderate
I/O Ports	Moderately fast	Moderately high
Buses	Moderate	Moderate
Controllers	Moderately slow	Moderate
Peripherals	Slow	Low
Humans	Very slow	Very high

- Spreading demand across resources will minimize potential bottlenecks.
- Identify the appropriate resource causing the bottleneck.

Tuning Strategies

Method	Cost	Ease to Implement	Impact on Performance
Changing system parameters	Cheap	Fairly easy	Minimal (unless extremely mistuned)
Changing process parameters	Cheap	Easy	Better
Adding hardware	Expensive	Moderate	Even better
Tuning application	Dependent on who wrote the application	Hard	Usually the best

OpenVMS Performance Metrics

Relative Units of Time

- Millisecond (ms)=.001 second
- Microsecond (us)=.000001 second
- Nanosecond (ns)=.000000001 second
- Picosecond (ps)=.000000000001 second
- CPU performance may be monitored through CPU time.
- A relatively complex activity like a page fault (100s of instructions) takes about 50 us on a 100MHz Alpha system.
- I/O performance is usually monitored through elapsed time.
- A disk I/O normally takes milliseconds to 10s of ms.
- Timings are affected by issues such as:
 - CPU caches
 - Virtual I/O cache
 - Controller caches

OpenVMS Performance Tools

Monitoring Tools

- MONITOR Utility
- \$SHOW SYSTEM
- \$SHOW STATUS
- \$SHOW MEMORY
- INSTALL Utility
- System Dump Analyzer (SDA)
- DUMP Utility
- Third-party products

Management Tools

- AUTHORIZE Utility
- AUTOGEN
- SYSMAN/SYSGEN
- INSTALL Utility
- RMS Utilities

Layout of Virtual Address Space

Module 2

Module Outline

Objectives

This module describes the layout of virtual address space to provide the conceptual foundations to understand future paging related discussions.

Topics

- Layout of Virtual Address Space
- P0 Space
- P1 Space
- S0/S1/S2 Space

Layout of Virtual Address Space

- OpenVMS/VAX systems support three virtual address spaces:
 - P0 space (program region)
 - P1 space (control region)
 - S0/S1 space (system region)
- P0 and P1 are per-process virtual address spaces.
 - Each process works with its own P0 and P1 space.
 - There is one copy of each address space for each process on the system.
- There is one copy of S0/S1 space, shared by all processes.
 - Code and data are required by all processes stored in S0 space.
 - Created during bootstrap of OpenVMS.
 - Remains relatively static through life of the system.
 - Some space pageable - charged against system's working set.

OpenVMS Alpha Memory Layout

00000000.00000000 to 00000000.3FFFFFFF	P0 Space
00000000.40000000 to 00000000.7FFFFFFF	P1 Space
00000000.80000000	P2 Space
Gap	
FFFFFFFB.FFFFFFFF	P2 Space
FFFFFFFC.00000000 to FFFFFFFD.FFFFFFFF	Page Table Space
FFFFFFFE.00000000 to FFFFFFF7.FFFFFFFF	S2 Space
FFFFFFF8.80000000 to FFFFFFF7.FFFFFFFF	S0/S1 Space

(Not to scale.)

Address Space

32-Bit Addressable Address Space Summary (VAX and Alpha)

Space	Region	Duration	Lifetime	Range
P0	Program	Image activation	Life of the image	0 to 3FFFFFFF
P1	Control	Process creation	Life of the process	40000000 to 7FFFFFFF
S0/S1	System	Bootstrap	Life of the system	80000000 to FFFFFFFF

64-Bit Addressable Address Space Summary (Alpha only)

Space	Region	Duration	Lifetime	Range
P2	64-bit program	Image activation	Life of the image	00000000. 80000000 to FFFFFFFB. FFFFFFFF
PT	Page table	Process creation	Life of the process	FFFFFFFC. 00000000 to FFFFFFFD. FFFFFFFF
S2	64-bit system	Bootstrap	Life of the system	FFFFFFFE. 00000000 to FFFFFFFF. 7FFFFFFF

OpenVMS Alpha Memory Management

Memory Terms

- Kilobyte (KB) 2^{10} (~1,000) bytes
- Megabyte (MB) 2^{20} (~1,000,000) bytes
- Gigabyte (GB) 2^{30} (~1,000,000,000) bytes
- Terabyte (TB) 2^{40} (~1,000,000,000,000) bytes
- Petabyte (PB) 2^{50} (~1,000,000,000,000,000) bytes

Address Space

- Variable page size: 8, 16, 32, or 64-KB compared to fixed 512 byte on VAX.
- 512 byte *pagelets* provide a translation mechanism to disk blocks and back to VAX architectures.
- Architectural limitation of 64-bit virtual and physical addresses.
- Alpha does not distinguish P0, P1, P2, S0, S1 and S2 space; OpenVMS does.

P0 Space Contents

The following describe the major components of P0 space:

Image

- Current image/program mapped into P0 space
- If no current image (that is, process is at DCL prompt), process has no P0 space
- P0 constantly reconstructed at each image activation , including most DCL commands (**TYPE**, **DIRECTORY**, **LINK**, etc.), implemented through images

Shareable Images

- Mapped into P0 space (for example, run-time libraries)
- Physical memory may be shared

Debugger and Traceback Handler

- Traceback handler displays detailed error message, including line number and calling sequence, when error status is returned from program

P1 Space Layout

The following describe some major components of P1 space:

Stacks

- Four per-process stacks (one per access mode) are stored in P1 space.

Command Language Interpreter (CLI)

- Interprets command lines entered by user and has its code and data mapped into P1 space.
- Typically DCL.

CLI Symbol Table

- CLI symbols (created by user) are stored in CLI symbol table in P1 space.
- Symbol table is sized by **SYSGEN** parameter **CLISYMTBL**.

Process Allocation Region

- Pool of memory set aside to store process data structures such as process logical names and image list of shareable images for image activator.
- Sized by **SYSGEN** parameter **CTL_PAGES**.

Channel Control Block Table

- Block allocated for each channel assigned by process.
- Table size limited by **SYSGEN** parameter **CHANNELCNT**.

RMS Buffers

- RMS prefetches disk blocks containing records into P1 space.
- Technique minimizes I/O activities.
- RMS buffer areas limited in size by **SYSGEN** parameters **PIOPAGES** and **IMGIOCNT**.

System Space Layout (Alpha)

S2 Space (Start: FFFFFFFE.00000000)

PFN Database
Permanent Mapping of System L1PT
Global Page Table (GPT)
Lock ID Table
Resource Hash Table
Lock Manager Pool Zone

S0/S1 Space (Start: FFFFFFFF.80000000)

Execlet Code Region
Resident Image Code Region
Execlet Data Region
System Header
Nonpaged Pool (initial size)
Nonpaged Pool Expansion Area
50 Balance Slots
Paged Pool
System Control Block (SCB)
Restart Parameter Block (HWRPB)
Swapper Process Kernel Stack
Swapper Map
Executive Mode Data Page
System Space Expansion Region

System Space

The following describe the major components of S0 space:

PFN Database

- Nonpageable portion of system space
- Contains the free, modified, and bad page lists, and other information required to track page table entries
- On Alpha systems the **PFN** database is allocated out of S2 space

Balance Slot Area

- Contains memory management data structures for processes called process headers (PHDs)
- Generally pageable and swappable, though working set list and process section table are locked into process' working set
- OpenVMS supports simultaneously memory-resident PHDs up to **SYSGEN** parameter **BALSETCNT**
- VAX systems support Virtual Balance Sets (see Appendix B)
- If more processes exist than **BALSETCNT + 2** (**Null** and **Swapper** do not use normal PHDs), then OpenVMS swaps processes to free up balance slots
- If plenty of memory, "artificial" swapping occurs and **BALSETCNT** should be increased
- Sample system that could perform "artificial" swapping:

\$ **show memory/slots**

```

System Memory Resources on 21-NOV-1999 06:34:26.97
Slot Usage (slots):          Total      Free      Resident      Swapped
  Process Entry Slots          30         0         29         1
  Balance Set Slots            19         1         18         0
$
```

System Working Set List

- Describes valid pages in pageable system space
- Pageable system space is charged against system working set list, except balance slots
- Sized by **SYSGEN** parameter **SYSMWCNT**

System Page Table

- Describes all of system space
- Nonpageable, and must be both virtually and physically contiguous (onVAX systems)
- Alpha systems only require virtual and physical contiguity at the page granularity.
- Size of system space affected by the majority of nondynamic **SYSGEN** parameters

Paged Pool

- Area set aside to store data structures accessed only from process context
- Examples include known file entries, global section descriptors, system/shared logical names and tables, file system caches
- Area is pageable and sized by **SYSGEN** parameter **PAGEDYN** in units of bytes
- Affects size of the system page table (nonpageable) and system's working set

S0/S1 Space

Nonpaged Pool

- Contains data structures touched from process and system context
- Pool is sized by **SYSGEN** parameter **NPAGEDYN**
- Fixed pool, sized by deallocation of packets into one of 128 (the VAX supports only 80 lookaside lists ranging from 64 to 5120 bytes in size) different sized lookaside lists, ranging from 64 to 8192 bytes in size
- Fixed packets minimize fragmentation and search time through variable pool when data structures conform to roughly the fixed size of packet
- If not enough pool, OpenVMS cannot perform necessary operations and hangs
- OpenVMS supports pool expansion allowing the size of these lists to grow by grabbing pages off free page list, up to **NPAGEVIR** bytes



Note

Paged and nonpaged pool are subsets of system space.

Monitoring Pool

```
$
$ show mem/pool
      System Memory Resources on 3-AUG-1998 08:02:32.00
Dynamic Memory Usage (bytes):      Total      Free      In Use      Largest
Nonpaged Dynamic Memory      3260416      98496      3161920      24384
Paged Dynamic Memory      2031616      1088336      943280      1087520
Lock Manager Dynamic Memory      966656      5632      961024

$
$
$ show mem/pool/full
      System Memory Resources on 3-AUG-1998 08:02:39.59
Nonpaged Dynamic Memory      (Lists + Variable)
  Current Size (bytes)      3260416      Current Size (pagelets)      6368
  Initial Size      2179072      Initial Size (pagelets)      4256
  Maximum Size      10895360      Maximum Size (pagelets)      21280
  Free Space (bytes)      98816      Space in Use (bytes)      3161600
  Largest Variable Block      16000      Smallest Variable Block      64
  Number of Free Blocks      135      Free Blocks LEQU 64 Bytes      13
  Free Blocks on Lookasides      74      Lookaside Space (bytes)      58176
(Bus Addressable Memory requests use Nonpaged Dynamic Memory)
Paged Dynamic Memory
  Current Size (PAGEDYN)      2031616      Current Size (pagelets)      3968
  Free Space (bytes)      1088336      Space in Use (bytes)      943280
  Largest Variable Block      1087520      Smallest Variable Block      16
  Number of Free Blocks      30      Free Blocks LEQU 64 Bytes      29
Lock Manager Dynamic Memory
  Current Size (Mbytes)      0.92      Current Size (pages)      118
  Free Space (Mbytes)      0.00      Hits      4007
  Space in Use (Mbytes)      0.91      Misses      56
  Number of Empty Pages      0      Expansions      118
  Number of Free Packets      21      Packet Size (bytes)      256

$
```



Note

If the current size of the nonpaged pool is greater than the initial size, the pool has expanded.

Nonpaged Pool Reclamation

- Fixed size pool packets that are deallocated are placed on the appropriate sized lookaside list.
- This behavior can cause depletion of the variable pool and can block allocation for requests that do not fit into available fixed-sized packets.
- To free up variable pool, OpenVMS performs pool reclamation in two flavors:

- **Gentle reclamation** (timer based)

Processes two of the 128 lists every **NPAG_INTERVAL** seconds. The default is 30 seconds, which means that it will take 32 minutes before all lists are processed.

Pool on the list is deallocated and then reallocated so that, original number of packets, the list is **NPAG_GENTLE** percent full. Defaults to 85%.

- **Aggressive reclamation** (allocation failure based)

Pool is reclaimed, up to **NPAG_AGGRESSIVE** (default 50%) full until one of the following:

- ◆ Request is satisfied.
- ◆ Request cannot be satisfied and pool is expanded.

SYSGEN> **SH NPAG**

Parameter Name	Current	Default	Min.	Max.	Unit	Dynamic
NPAGEDYN	1474560	1048576	163840		-1 Bytes	
NPAGEVIR	7372800	8388608	163840		-1 Bytes	
NPAG_INTERVAL	30	30	0		-1 Seconds	D
NPAG_GENTLE	85	85	0	100	Percent	D
NPAG_AGGRESSIVE	50	50	0	100	Percent	D
NPAG_BAP_MIN	0	0	0		-1 Bytes	
NPAG_BAP_MAX	0	0	0		-1 Bytes	
NPAG_BAP_MAX_PA	-1	-1	0		-1 PhysAddr	
NPAG_RING_SIZE	2048	2048	0		-1 Entries	

SYSGEN>

Make sure all lookaside lists are processed every 128 seconds.

SYSGEN> **SET NPAG_INT 2**
 SYSGEN> **WRITE ACT**
 SYSGEN>
 SYSGEN>
 SYSGEN> **EXIT**
 \$ **ANA/SYS**
 OpenVMS (TM) Alpha system analyzer

SDA>

Illustration of lookaside lists.

SDA> **clue mem/look**

Nonpaged Dynamic Storage Pool - Lookaside List Queue Information:

Listhead Addr: FFFFFFFF.80C28C00	Size: 64	Status: Valid, 3 elements
Listhead Addr: FFFFFFFF.80C28C08	Size: 128	Status: Valid, 2 elements
Listhead Addr: FFFFFFFF.80C28C10	Size: 192	Status: Valid, 1 element
Listhead Addr: FFFFFFFF.80C28C18	Size: 256	Status: Valid, 1 element
Listhead Addr: FFFFFFFF.80C28C20	Size: 320	Status: Valid, empty
Listhead Addr: FFFFFFFF.80C28C28	Size: 384	Status: Valid, 1 element
Listhead Addr: FFFFFFFF.80C28C30	Size: 448	Status: Valid, 1 element
Listhead Addr: FFFFFFFF.80C28C38	Size: 512	Status: Valid, 1 element
Listhead Addr: FFFFFFFF.80C28C40	Size: 576	Status: Valid, 5 elements
Listhead Addr: FFFFFFFF.80C28C48	Size: 640	Status: Valid, 2 elements
Listhead Addr: FFFFFFFF.80C28C50	Size: 704	Status: Valid, 1 element
Listhead Addr: FFFFFFFF.80C28C58	Size: 768	Status: Valid, 1 element
Listhead Addr: FFFFFFFF.80C28C60	Size: 832	Status: Valid, 2 elements
Listhead Addr: FFFFFFFF.80C28C68	Size: 896	Status: Valid, 1 element
Listhead Addr: FFFFFFFF.80C28C70	Size: 960	Status: Valid, 1 element
Listhead Addr: FFFFFFFF.80C28C78	Size: 1024	Status: Valid, 1 element
Listhead Addr: FFFFFFFF.80C28C80	Size: 1088	Status: Valid, 1 element
Listhead Addr: FFFFFFFF.80C28C88	Size: 1152	Status: Valid, 1 element
Listhead Addr: FFFFFFFF.80C28C90	Size: 1216	Status: Valid, empty

Press RETURN for more.

SDA>

Force packets onto lookaside list.

SDA> **spawn r dka300:[ellis]many_timers**

SDA> **clue mem/look**

Nonpaged Dynamic Storage Pool - Lookaside List Queue Information:

Listhead Addr: FFFFFFFF.80C28C00 Size: 64 Status: Valid, 104 elements
Listhead Addr: FFFFFFFF.80C28C08 Size: 128 Status: Valid, 4 elements
Listhead Addr: FFFFFFFF.80C28C10 Size: 192 Status: Valid, 7 elements
Listhead Addr: FFFFFFFF.80C28C18 Size: 256 Status: Valid, 1 element

...

Press RETURN for more.

SDA> **clue mem/look**

Nonpaged Dynamic Storage Pool - Lookaside List Queue Information:

Listhead Addr: FFFFFFFF.80C28C00 Size: 64 Status: Valid, 75 elements
Listhead Addr: FFFFFFFF.80C28C08 Size: 128 Status: Valid, 3 elements
Listhead Addr: FFFFFFFF.80C28C10 Size: 192 Status: Valid, 6 elements
Listhead Addr: FFFFFFFF.80C28C18 Size: 256 Status: Valid, 1 element

...

Press RETURN for more.

Force less gentle reclamation.

```
SDA> spawn mcr sysgen
SYSGEN> SET NPAG_GENT 25
SYSGEN> WRITE ACT
SYSGEN> exit
SDA> clue mem/look
Nonpaged Dynamic Storage Pool - Lookaside List Queue Information:
-----
Listhead Addr: FFFFFFFF.80C28C00   Size:   64   Status: Valid, 20 elements
Listhead Addr: FFFFFFFF.80C28C08   Size:  128   Status: Valid,  3 elements
Listhead Addr: FFFFFFFF.80C28C10   Size:  192   Status: Valid,  9 elements
Listhead Addr: FFFFFFFF.80C28C18   Size:  256   Status: Valid,  1 element
...

Press RETURN for more.
SDA> clue mem/look
Nonpaged Dynamic Storage Pool - Lookaside List Queue Information:
-----
Listhead Addr: FFFFFFFF.80C28C00   Size:   64   Status: Valid,  6 elements
Listhead Addr: FFFFFFFF.80C28C08   Size:  128   Status: Valid,  1 element
Listhead Addr: FFFFFFFF.80C28C10   Size:  192   Status: Valid,  3 elements
Listhead Addr: FFFFFFFF.80C28C18   Size:  256   Status: Valid,  1 element
...

Press RETURN for more.
SDA> clue mem/look
Nonpaged Dynamic Storage Pool - Lookaside List Queue Information:
-----
Listhead Addr: FFFFFFFF.80C28C00   Size:   64   Status: Valid,  3 elements
Listhead Addr: FFFFFFFF.80C28C08   Size:  128   Status: Valid,  1 element
Listhead Addr: FFFFFFFF.80C28C10   Size:  192   Status: Valid,  4 elements
Listhead Addr: FFFFFFFF.80C28C18   Size:  256   Status: Valid,  1 element
...

Press RETURN for more.
SDA>
```

\$ sh mem/pool/full

System Memory Resources on 3-SEP-1997 11:26:18.44

Nonpaged Dynamic Memory (Lists + Variable)

Current Size (bytes)	1507328	Current Size (pagelets)	2944
Initial Size	1474560	Initial Size (pagelets)	2880
Maximum Size	7372800	Maximum Size (pagelets)	14400
Free Space (bytes)	89664	Space in Use (bytes)	1417664
Largest Variable Block	17920	Smallest Variable Block	64
Number of Free Blocks	143	Free Blocks LEQU 64 Bytes	33
Free Blocks on Lookasides	39	Lookaside Space (bytes)	40384

(No Bus Addressable Memory allocated)

Paged Dynamic Memory

Current Size (PAGEDYN)	622592	Current Size (pagelets)	1216
Free Space (bytes)	183840	Space in Use (bytes)	438752
Largest Variable Block	182368	Smallest Variable Block	16
Number of Free Blocks	44	Free Blocks LEQU 64 Bytes	42

\$

Force pool expansion.

\$ mcr sysgen

SYSGEN> SET NPAG_AGG 100

SYSGEN> WRITE ACT

SYSGEN> Exit

\$ set host 0

Welcome to OpenVMS (TM) Alpha Operating System, Version G7.1

Username: Exit

Error reading command input

End of file detected

%REM-S-END, control returned to node LOCAL:.ELLIS::

\$ sh mem/pool/full

System Memory Resources on 3-SEP-1997 11:35:28.36

Nonpaged Dynamic Memory (Lists + Variable)

Current Size (bytes)	1572864	Current Size (pagelets)	3072
Initial Size	1474560	Initial Size (pagelets)	2880
Maximum Size	7372800	Maximum Size (pagelets)	14400
Free Space (bytes)	151040	Space in Use (bytes)	1421824
Largest Variable Block	82880	Smallest Variable Block	64
Number of Free Blocks	167	Free Blocks LEQU 64 Bytes	31
Free Blocks on Lookasides	102	Lookaside Space (bytes)	59648

(No Bus Addressable Memory allocated)

Paged Dynamic Memory

Current Size (PAGEDYN)	622592	Current Size (pagelets)	1216
Free Space (bytes)	183840	Space in Use (bytes)	438752
Largest Variable Block	182368	Smallest Variable Block	16
Number of Free Blocks	44	Free Blocks LEQU 64 Bytes	42

\$



Notes on Reclamation

It is expensive on allocation and can cause spinwait time out crashes in an SMP environment.

You can force expansion, which is usually cheaper in the short term, by setting NPAG_AGGRESSIVE to 100. The long-term cost will be potential depletion of the free page list.

More frequent gentle reclamation may free up enough space for allocations, at a cost of interrupt state time.

Pool Zones

- Pool zones were introduced in V7.2 to take some of the burden from nonpaged pool.
- Pool zones work in some ways very much like nonpaged pool.
 - Non-pageable
 - Not guaranteed to be physically contiguous
 - Hold data structures
- Pool zones are different from nonpaged pool in other ways.
 - Operate on pages
 - Can dynamically expand and contract
 - No system parameters to limit size
 - Not guaranteed to be virtually contiguous
 - Synchronization provided by caller
 - Deal with single-sized structures per pool zone
 - Managed by a single type of code
 - Allocated out of S2 space (minimum of 6GB scalable through the **S2_SIZE SYSGEN** parameter)
 - Currently used only by the Lock Manager, but multiple pool zones are possible

Lock Manager Use of Pool Zones

- Used to hold only RSBs and LKBs in 256 byte packets held within a pool zone page.
- Up to 31 packets per page.
- Minimizes fragmentation of nonpaged pool.
- Separate synchronization.
- Cost is 30% greater in memory usage for lock management structures than old pool mechanism.
- **LOCKIDTBL** **SYSGEN** parameter may be used to guarantee minimum memory held by Lock Manager.
- If all 31 packets in a page are unused, it may be returned to the Free Page List.
- May be monitored using **\$SHOW MEMORY** or in SDA.
- No SYSGEN parameters to worry about. (May affect initial allocation and lower bound, using the parameter **LOCKIDTBL**).

Image Activation and Paging

Module 3

Module Outline

Objectives

This module describes image activation as it relates to paging and paging metrics. Upon completion of this module, students will be able to explain the process of image activation and the types of page faults that are incurred along the way. Students should be able to distinguish required page faults and those that are overhead.

Topics

- Page Faults
- Producing an Image File
- Image Activation and Paging
- Valid Pages and the Free and Modified Page Lists
- The Free List Fault Rate
- The Modified List Fault Rate
- Secondary Hard Faults

Paging Compared to Swapping

Paging

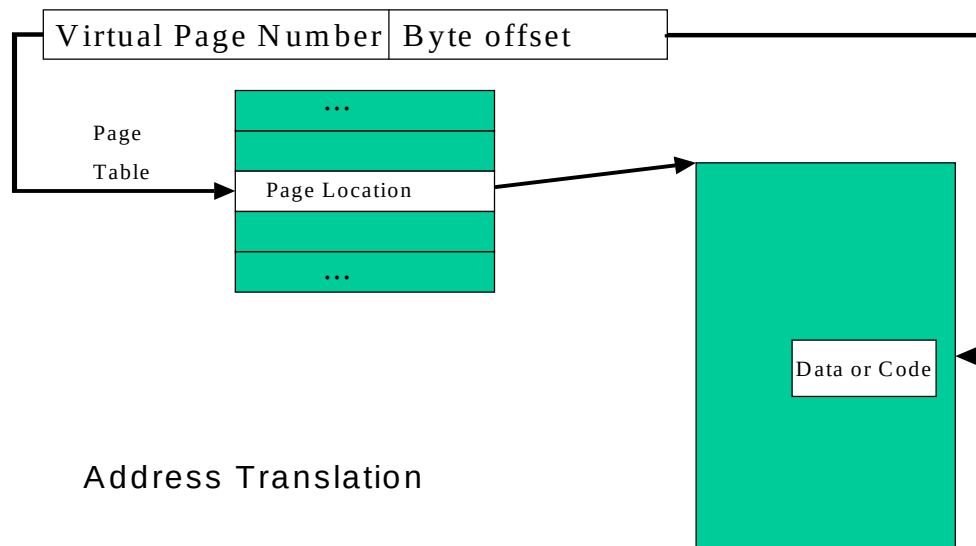
- Supports programs larger than memory can hold
- Is a process activity
- Process is charged for page faults incurred and CPU time expended to resolve page faults
- Execution does not require entire program be memory resident simultaneously

Swapping

- Supports more processes than memory can hold
- System-wide activity—effort to reclaim memory for processes
- CPU time and other performance metrics charged against **SWAPPER** process

Address Translation

- Every instruction that is executed must be located by its address.
- All data accessed by an instruction must be located by its address.



Address Translation Model

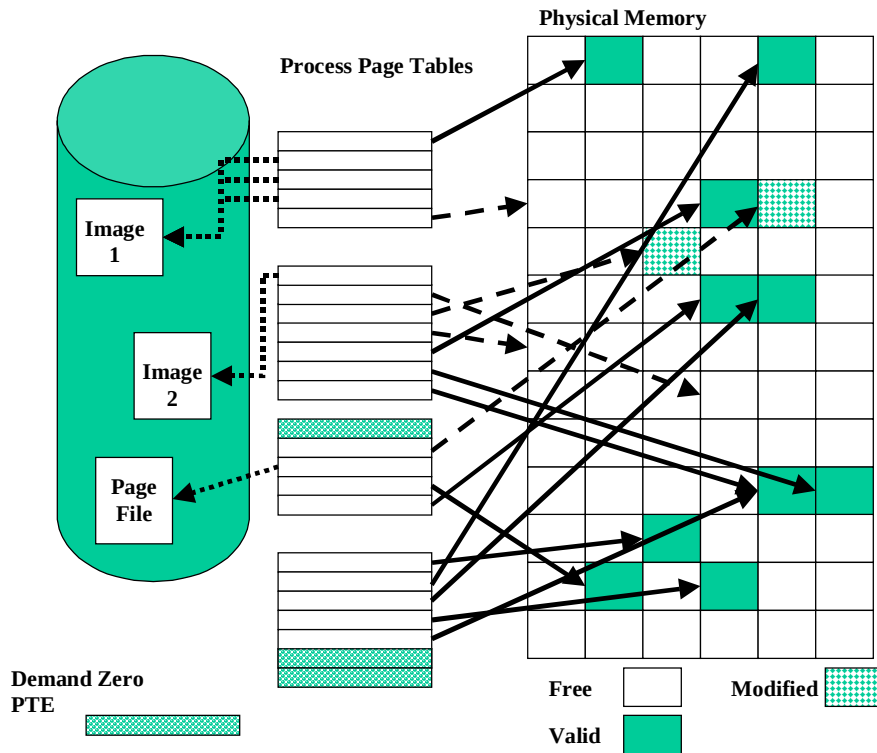
- Details of address translation vary between VAX and Alpha systems.

Page Faults

The formal name of a page fault is an *address translation not valid exception*.

- Page faults make pages valid.
- Paging is a natural progression in activation of an image.
- A common paging myth is that a performance goal should be to eliminate or minimize page faults.
- It is important to understand the quality of faults incurred and whether they are avoidable or a natural part of activation of images.
- Remember, page faults are not inherently bad, from a performance viewpoint.
- *Soft fault* is resolved by **CPU** in memory.
- *Hard fault* requires a disk I/O operation to resolve.

Locating Pages in Virtual Memory



- Processes all have private page tables.
- When a process is currently executing, its page table is in use.
- Pages may be located in one of the many ways illustrated above. (The diagram does not include global paging.) Solid lines indicate that no page fault is required. Dashed lines indicate soft faults. Dotted lines indicate hard faults.

Producing an Image File

An image is a set of code or data generated by linker. An executable image or program can be run.

When writing a program in a high-level language, attributes are designated for data and code (describe run-time characteristics of code or data).

- Attributes include read-only compared to writable; executable compared to non-executable; shared compared to non-shared, and so on.
- In a high-level language, attributes are implied by programming constructs and applied by compiler.
- In MACRO, you can assign these attributes with **.PSECT** directive.

Data/code compiled or assembled is organized in an object module in *sections* that have been assigned these attributes. For example, writable data would go into a different section from read-only code.

Using the **\$LINK** command produces executable image made up of three parts: *image header*, *image body*, and *fixup vector*.

Image Header

- Contains address of location at which program will start
- Also contains image section descriptors (ISDs) describing sections of code and the data image body
- Information stored in ISDs includes:
 - Starting virtual block number of code/data in image file
 - Size of section in pages
 - Starting virtual page number of section in memory
 - Flags describing attributes of section

Image Body

Contains binary description of sections of data and code that have been stored in the image file.



Note

Not all data will be stored in the image body—pages of uninitialized data go into demand zero sections (which have image section descriptors in the image header) but no representation in the image body.

Fixup Vector

Contains information describing references to shareable images allowing position-independent access to shareable images that this image has linked against, regardless of where they are mapped to in virtual address space.

Image Activation and Paging

The image activator reads in the image header from the image file. It uses information to construct page table entries and process section table entries.

Process Section Table Entries

- Used to locate sections of code and data in image file
- Contain similar information to image section descriptors in image file, but are memory resident
- Information in process section table entry includes:
 - Starting virtual block number in image file of code/data
 - Size of section in pages
 - Starting virtual page number of page in memory
 - Flags describing attributes of section
 - Channel number for open image file
 - Pointer to a window structure allowing pager to perform virtual to logical block mapping
- All of this information is necessary to perform any required paging I/O on a hard page fault.

Page Table Entries

- Image activator constructs 3 initial forms of page table entry contents at image activation:
 - Process section table index (**PSTX**)
 - Demand zero entry (**DZRO**)
 - Global page table index (**GPTX**)

Process Section Table Index

The pager will initiate a hard fault on access.

- Due to prefetching, one or more pages from the same section, up to a default of **PFCDEFAULT**, will be read in and made valid.
- This will display as **PAGE READ I/O** (hard fault) in **\$MONITOR PAGE**.
- To determine amount of prefetching, examine **PAGE READ** in **\$MONITOR PAGE** (describes number of pages being made valid due to hard faults per second).
- Dividing the **PAGE READ** rate by the **PAGE READ I/O** rate indicates the prefetch rate or how many pages are being made valid due to hard faults.
- Hard faults are generally required as part of the image activation process.
- To minimize these faults, consider minimizing the image activations.

Demand Zero Entry

For uninitialized pages of data, the linker generates an image section descriptor that describes these pages as being demand zero, but no space is allocated for these pages in the image file.

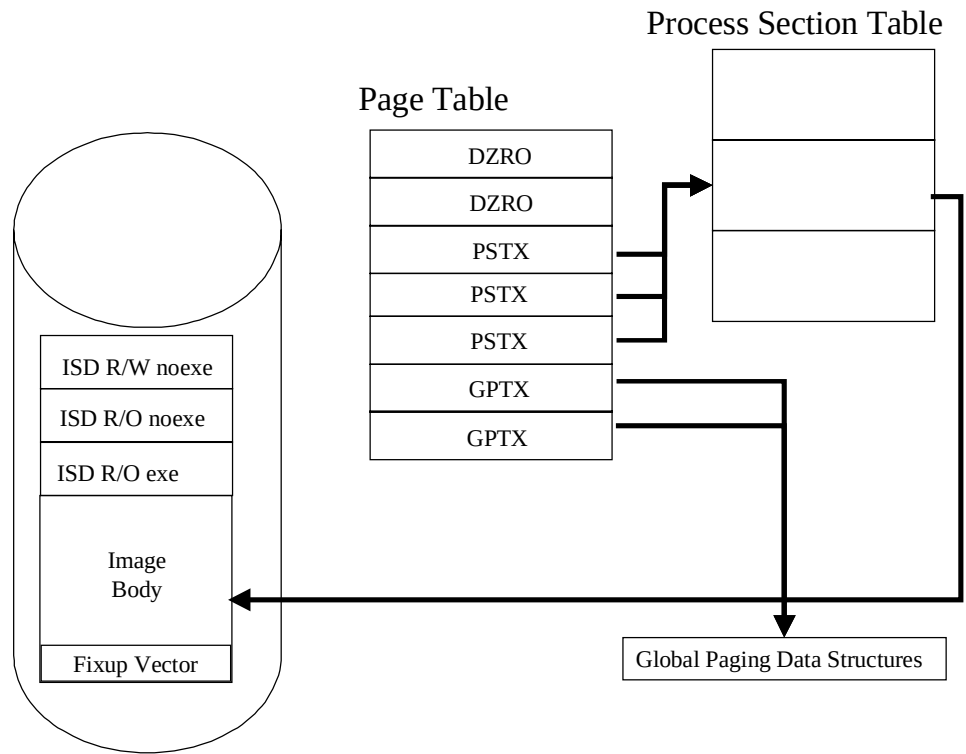
- When one of these pages is accessed, a soft page fault is generated.
- When the pager detects entry is for demand zero page, a page is taken from the free page list, made valid, and zeroed with one instruction (on a VAX). On an Alpha system, pages are zeroed by the idle loop in the scheduler and taken from the zeroed list, if available. Up to **ZERO_LIST_HI** (**SYSGEN** parameter) pages may be on this list.
- These page faults occur once in the life of the page.
- Demand zero faults may indicate that image activations are occurring.
- Nothing you can do will eliminate these faults other than eliminating image activations occurring.
- The paging increase will be due to the result of faster image activation and is not inherently bad, but rather good because the image activation process is occurring faster.
- These faults would show up as **DEMAND ZERO FAULTS** in **\$MONITOR PAGE**.

Global Page Table Index

These page table entries are established for global pages.

- These entries are associated with images installed **/SHARE** and typically refer to read-only pages (although data may be shared by using shared data within shareable image or through create and map section (**\$CRMPSC**) system service).
- When these pages are accessed, a page fault occurs.
- To resolve the fault, the pager will use the global page table index to locate the global page table entry, which will identify the current location of global page.
- The global page may not be memory resident, in which case, the pager will generate a hard page fault (**PAGE READ I/O**).
- The page may already be valid for another process and a soft fault may be generated (**GOBAL VALID FAULT**).
- The page may be in memory, on the free page list, from a previous run of the program (**FREE LIST FAULT**).

Image Activation



Free Page List

- The free page list is a list of pages available for processes that need physical pages to be added to the working set.
- Processes require these pages to resolve hard faults and demand zero faults.



Note

The list describing the pages is distinct from pages themselves.

- As processes make more pages valid, the free list shrinks in size.
- To limit the number of pages a process may own, valid pages are charged against the process' *working set list*.

Valid Pages and the Working Set List

Working Set List

- Acts as a memory limit for processes.
- As more pages become valid for process, the list fills.
- After the list is full, a page from the working set list must be displaced before another can be made valid.
- The page replacement mechanism on OpenVMS is fundamentally first-in-first-out (FIFO).
- Though fairly simplistic, it has minimal overhead to maintain.
- If a heavily used page is displaced from the working set list, it is not lost from memory and may have a later fault resolved without an I/O.
- Content of a page table entry for a valid page has the valid bit set and a page frame number (**PFN**).
- The **PFN** identifies the physical location of the page in memory.

Displacing a Page from the Working Set List

- When a page is displaced from the working set list, its page table entry has one of two contents: **PFN** with valid bit clear or global page table index.
- If **PFN** in the page table entry with valid bit clear, the page is either on the free or modified page list.

Free List Fault Rate

Faulting a page from the free list is a soft fault. It is more desirable to keep a page in the working set and never have to fault on the page at all.

- A high **FREE LIST FAULT** rate in **\$MONITOR PAGE** may indicate the working set lists are sized too small.
- If working set quotas are sized properly (to avoid over-committing memory), some free list faults may be minimized by setting larger working set extents.



Note

FREE LIST FAULT rate may also increase because global pages, which still reside in memory, are being faulted from the free page list.

- It is difficult to determine whether a high **FREE LIST FAULT** rate is good (faster image activation due to global pages that are still in memory), or bad (too small a working set).
- If **DEMAND ZERO FAULT** and **GLOBAL VALID FAULT** rates are also going up, then image activations are probably causing the higher free list fault rate.
- You may consider minimizing the number of image activations by rewriting command procedures as programs.

Modified List Fault Rate

The modified list fault rate functions like the free page list, as it describes pages displaced from working set lists.

The fundamental difference is that pages on the modified page list have changed since they were first accessed and have no representation on disk.

- Pages could have been written to disk immediately upon displacement from the working set list.
 - However, this would lead to many small and potentially unnecessary I/Os.
 - Instead, they remain on the modified page list until faulted back into the working set by process, or until the page is written to page file.
- Pages are written to page file when the list is too full, or possibly when the free list gets too small.
- Pages on the modified page list may be faulted off by soft fault (*modified page list fault*) in a way similar to pages faulted from the free list.
- Modified page list faults are undesirable because pages being faulted off the list would have been used more effectively if they had never left the process working set.
 - It may not be reasonable to increase the process' working set quotas to a large degree because this is a memory commitment to the process.
- High modified list fault rate, therefore, may be an indication to review working set quotas and memory availability.

Secondary Hard Faults

When a page is taken from the top of the free page list by the process not interested in the page, the pager must update the process page table entry (if there is one) with information allowing the page to be located on disk. These contents may be the page file virtual block number or section table index.

Process Section Table Index

- Placed back into page table entry when the page is still stored in its original form in the image file (although non-copy-on-reference or shareable writable pages within a shareable image may have been written back to image file).
- A process attempt to access the page generates a hard fault (**PAGE READ I/O**) to make it valid.

Page File Virtual Block Number

- If a page contains writable data, it is typically written to the page file prior to being placed on the free page list.
- When a process later attempts to access the page, it generates a hard fault (**PAGE READ I/O**) to make the page valid.

Secondary Hard Faults

- These faults are secondary hard faults (though OpenVMS does not report them as such).
- Different from hard faults at image activation in that they are not required by the image activation process.
- Caused by the free list being small enough to lose a page from memory entirely.
- Unlike hard faults at image activation, these faults can and should be avoided.
- Though OpenVMS does not differentiate secondary hard faults, generally assume hard faulting is due to secondary hard faults if:
 - The free list is relatively small, **PAGE READ I/O RATE** is high.
 - Processes start showing up in **PFW** (page fault wait) state for extended periods of time.
- If the hard fault rate is high, consider several actions, including:
 - Determine whether you have over-committed your working set quotas. Do they add up to more than you have the memory to support.
 - Do you need more memory? This type of hard faulting is the primary reason to purchase more memory.
 - Can you limit the number of concurrent users and force large jobs into batch queues with larger working set sizes?

Interpreting Paging Metrics

Module 4

Module Outline

Objectives

This module walks through several paging examples to display how varying workloads will affect paging metrics. It also summarizes the general steps involved in interpreting paging metrics.

Topics

- Sample Display from \$MONITOR Page
- Interpreting the \$MONITOR Page Display
- Interpreting Paging Metrics

Sample Display from \$Monitor Page

```

VAX/VMS Monitor Utility
PAGE MANAGEMENT STATISTICS
  on node BRUDEN
 24-OCT-1999 11:50:57

              CUR      AVE      MIN      MAX

Page Fault Rate      2.33      2.33      2.33      2.33
Page Read Rate       6.00      6.00      6.00      6.00
Page Read I/O Rate   0.66      0.66      0.66      0.66
Page Write Rate      0.00      0.00      0.00      0.00
Page Write I/O Rate  0.00      0.00      0.00      0.00

Free List Fault Rate  1.33      1.33      1.33      1.33
Modified List Fault Rate 0.00      0.00      0.00      0.00
Demand Zero Fault Rate 0.33      0.33      0.33      0.33
Global Valid Fault Rate 0.00      0.00      0.00      0.00
Wrt In Progress Fault Rate 0.00      0.00      0.00      0.00
System Fault Rate    0.00      0.00      0.00      0.00

Free List Size      4292.00    4292.00    4292.00    4292.00
Modified List Size   491.00     491.00     491.00     491.00
  
```

Interpreting the \$Monitor Page Display

Page Fault Rate

- Identifies overall page fault rate on system, including hard and soft faults.

Read I/O Rate

- Identifies hard fault rate system wide.
- High hard fault rate indicates memory exhaustion or possibly high image activation rate.

Page Read Rate

- Identifies how many pages are being made valid due to hard faults.
- Dividing this rate by **Page Read I/O Rate** indicates how much pre-fetching is being done per hard fault.

Page Write I/O Rate

- These are not page faults.
- This is the number of times clusters of pages are being written to page file per second.

Page Write Rate

- Identifies the number of pages written to page file per second.
- Dividing **Write Rate** by **Write I/O Rate** roughly yields setting for **MPW_WRTCLUSTER** on system (if not, modified page writing is inefficient).

Free List Fault Rate

- Defines number of soft faults from free page list per second.
- High free list fault could be indication that working set lists are too small.
- Could also indicate performing more efficient image activations due to faulting global pages from free list.

Modified List Fault Rate

- Identifies number of soft faults resolved from modified page list per second.
- If high, typically indicates working set lists sized too small.

Demand Zero Fault Rate

- Identifies number of soft page faults resolved on uninitialized data pages.
- May be associated with excessive image activation.

Global Valid Fault Rate

- Identifies number of page faults resolved through global pages already valid in memory.
- High rate may indicate working set lists are too small.
- May also indicate high image activation rate.

Wrt-In-Progress Fault Rate

- Indicates number of faults incurred on pages **SWAPPER** was in process of writing to page file per second.
- If pages were faulted off **SWAPPER**'s **I/O** map, the pages should not have been written at all.
- Rate is an indication of inefficient modified page writing.

System Fault Rate

- Faults not charged against any process and are on pageable system space other than balance slot area.
- Balance system working set size based on actual system paging.
- May free up memory for process paging usage.
- To eliminate these faults, set parameter **SYSMWCNT** properly and turn off system **AWSA**.

Free List Size

- Indicates spare memory available to process working set lists to minimize soft faulting.
- Do not increase working set list sizes if free list size is low.
- If the free list is small, consider decreasing the working set list sizes.

Modified List Size

- Indicates current size of modified page list.

Interpreting Paging Metrics

VAX/VMS Monitor Utility
 PAGE MANAGEMENT STATISTICS
 on node BRUDEN
 1-AUG-1999 16:45:35

	CUR	AVE	MIN	MAX
Page Fault Rate	4771.47	4771.45	4771.42	4771.47
Page Read Rate	0.32	2.58	0.32	4.76
Page Read I/O Rate	0.32	0.32	0.31	0.32
Page Write Rate	0.00	0.00	0.00	0.00
Page Write I/O Rate	0.00	0.00	0.00	0.00
Free List Fault Rate	4.26	5.32	4.26	6.34
Modified List Fault Rate	4766.55	4765.00	4763.49	4766.55
Demand Zero Fault Rate	0.32	0.48	0.32	0.63
Global Valid Fault Rate	0.00	0.32	0.00	0.63
Wrt In Progress Fault Rate	0.00	0.00	0.00	0.00
System Fault Rate	0.00	0.00	0.00	0.00
Free List Size	2205.00	2203.50	2202.00	2205.00
Modified List Size	639.00	641.00	639.00	643.00

- In this display, note that the page fault rate is fairly high.
 - Also note majority of faults are *modified list faults*.
 - Indicates working set list size for process is too small.
 - If the working set list size were higher, pages would not have made it to the modified page list in first place.
- You may not be able to size the working set list quota large enough to support the processes that are paging, due to the potential demand.
 - To minimize some of these faults when the load may vary, consider larger working set extent.

VAX/VMS Monitor Utility
PAGE MANAGEMENT STATISTICS
on node BRUDEN
1-AUG-1999 16:46:44

	CUR	AVE	MIN	MAX
Page Fault Rate	3945.74	3997.96	3862.09	4148.91
Page Read Rate	23.34	24.93	23.27	28.10
Page Read I/O Rate	23.34	23.28	21.67	24.83
Page Write Rate	37.85	38.29	37.15	39.86
Page Write I/O Rate	0.31	0.38	0.30	0.65
Free List Fault Rate	13.56	13.86	12.89	15.78
Modified List Fault Rate	3907.25	3958.52	3819.60	4108.97
Demand Zero Fault Rate	0.00	0.19	0.00	0.61
Global Valid Fault Rate	0.00	0.00	0.00	0.00
Wrt In Progress Fault Rate	0.00	0.12	0.00	0.65
System Fault Rate	0.00	0.00	0.00	0.00
Free List Size	137.00	127.80	119.00	137.00
Modified List Size	603.00	613.00	603.00	625.00

- This display was generated by running the program from the previous example under three processes.



Note

Although the page fault rate has gone down, paging statistics indicate poorer performance.

- *Page read I/O rate* has gone up to 23.34 per second.
 - Indicates processes are generating more hard faults.
 - Primarily because the free list size is small (127.80 pages avg.).
 - Processes are now exhausting memory.
 - To minimize impact, consider purchasing more memory, lowering working set quotas, or restricting the number of concurrent runs of the program, through batch stream.

VAX/VMS Monitor Utility
PAGE MANAGEMENT STATISTICS
on node BRUDEN
1-AUG-1999 16:47:46

	CUR	AVE	MIN	MAX
Page Fault Rate	363.33	387.31	359.81	439.62
Page Read Rate	18.48	16.81	14.01	21.81
Page Read I/O Rate	16.96	15.33	14.01	16.96
Page Write Rate	356.36	366.81	338.94	415.09
Page Write I/O Rate	3.63	3.38	3.11	3.63
Free List Fault Rate	3.33	1.66	0.00	3.33
Modified List Fault Rate	4.24	3.57	2.80	4.24
Demand Zero Fault Rate	0.00	0.18	0.00	0.60
Global Valid Fault Rate	2.12	0.80	0.00	2.12
Wrt In Progress Fault Rate	317.57	347.16	317.57	400.00
System Fault Rate	0.00	0.00	0.00	0.00
Free List Size	557.00	566.79	557.00	574.00
Modified List Size	464.00	520.79	464.00	535.00

- In this display, the system started to go beyond the ragged edge of exhausting memory.
 - Page fault rate has dropped substantially.
 - Now writing modified page list 10 times as frequently.
 - Faults being resolved include hard faults (Page Read I/O rate) and *Wrt-In-Progress* faults.
- Write-in-progress faults are faults of pages that **SWAPPER** is attempting to write to page file.
 - Indicates inefficient modified page writing.

Paging Metrics Alpha vs. VAX

The interpretation of paging metrics on Alpha systems is fundamentally the same as that of VAX systems.

Following are some general differences in appearance:

- The page fault rate will burst at higher rates on Alpha due to CPU speed.
- Prefetching will appear lower on Alpha systems due to a larger page size.
- Granularity of the Free and Modified page lists, on Alpha systems, is 16 times larger than that of VAX systems.
- Dependent on the behavior of the application, total page faults may be less on Alpha systems than VAX systems.

OpenVMS Monitor Utility PAGE MANAGEMENT STATISTICS on node ELLIS 5-SEP-1999 09:26:39.18				
	CUR	AVE	MIN	MAX
Page Fault Rate	8012.97	8012.97	8012.97	8012.97
Page Read Rate	15.36	15.36	15.36	15.36
Page Read I/O Rate	7.18	7.18	7.18	7.18
Page Write Rate	19.36	19.36	19.36	19.36
Page Write I/O Rate	0.99	0.99	0.99	0.99
Free List Fault Rate	45.10	45.10	45.10	45.10
Modified List Fault Rate	7854.29	7854.29	7854.29	7854.29
Demand Zero Fault Rate	99.80	99.80	99.80	99.80
Global Valid Fault Rate	6.18	6.18	6.18	6.18
Wrt In Progress Fault Rate	0.00	0.00	0.00	0.00
System Fault Rate	0.00	0.00	0.00	0.00
Free List Size	96.00	96.00	96.00	96.00
Modified List Size	491.00	491.00	491.00	491.00

Alpha \$MONITOR Page

PAGE MANAGEMENT STATISTICS
on node ELLIS
5-SEP-1999 09:26:42.65

	CUR	AVE	MIN	MAX
Page Fault Rate	29318.20	16716.17	8012.97	29318.20
Page Read Rate	29.19	21.01	15.36	29.19
Page Read I/O Rate	12.42	9.32	7.18	12.42
Page Write Rate	18.49	19.00	18.49	19.36
Page Write I/O Rate	0.28	0.70	0.28	0.99
Free List Fault Rate	147.10	86.77	45.10	147.10
Modified List Fault Rate	29147.10	16552.41	7854.29	29147.10
Demand Zero Fault Rate	0.57	59.26	0.57	99.80
Global Valid Fault Rate	10.69	8.02	6.18	10.69
Wrt In Progress Fault Rate	0.00	0.00	0.00	0.00
System Fault Rate	0.00	0.00	0.00	0.00
Free List Size	93.00	94.50	93.00	96.00
Modified List Size	377.00	434.00	377.00	491.00

OpenVMS Monitor Utility
PAGE MANAGEMENT STATISTICS
on node ELLIS
5-SEP-1999 09:26:45.75

	CUR	AVE	MIN	MAX
Page Fault Rate	31770.32	20731.77	8012.97	31770.32
Page Read Rate	3.87	16.40	3.87	29.19
Page Read I/O Rate	0.96	7.08	0.96	12.42
Page Write Rate	0.00	13.90	0.00	19.36
Page Write I/O Rate	0.00	0.51	0.00	0.99
Free List Fault Rate	136.77	100.08	45.10	147.10
Modified List Fault Rate	31632.25	20575.04	7854.29	31632.25
Demand Zero Fault Rate	0.00	43.35	0.00	99.80
Global Valid Fault Rate	0.32	5.95	0.32	10.69
Wrt In Progress Fault Rate	0.00	0.00	0.00	0.00
System Fault Rate	0.00	0.00	0.00	0.00
Free List Size	72.00	87.00	72.00	96.00
Modified List Size	367.00	411.66	367.00	491.00

OpenVMS Monitor Utility
PAGE MANAGEMENT STATISTICS
on node ELLIS
5-SEP-1999 09:26:48.76

	CUR	AVE	MIN	MAX
Page Fault Rate	31055.66	22856.03	8012.97	31770.32
Page Read Rate	0.00	13.03	0.00	29.19
Page Read I/O Rate	0.00	5.62	0.00	12.42
Page Write Rate	21.33	15.43	0.00	21.33
Page Write I/O Rate	0.33	0.48	0.00	0.99
Free List Fault Rate	130.33	106.31	45.10	147.10
Modified List Fault Rate	30925.33	22704.73	7854.29	31632.25
Demand Zero Fault Rate	0.00	34.43	0.00	99.80
Global Valid Fault Rate	0.00	4.73	0.00	10.69
Wrt In Progress Fault Rate	0.00	0.00	0.00	0.00
System Fault Rate	0.00	0.00	0.00	0.00
Free List Size	111.00	93.00	72.00	111.00
Modified List Size	303.00	384.50	303.00	491.00

OpenVMS Monitor Utility

PAGE MANAGEMENT STATISTICS
on node ELLIS
5-SEP-1999 09:26:51.76

	CUR	AVE	MIN	MAX
Page Fault Rate	28676.00	23835.64	8012.97	31770.32
Page Read Rate	42.33	18.02	0.00	42.33
Page Read I/O Rate	19.00	7.90	0.00	19.00
Page Write Rate	0.00	12.79	0.00	21.33
Page Write I/O Rate	0.00	0.39	0.00	0.99
Free List Fault Rate	138.66	111.76	45.10	147.10
Modified List Fault Rate	28504.00	23680.89	7854.29	31632.25
Demand Zero Fault Rate	0.00	28.53	0.00	99.80
Global Valid Fault Rate	14.00	6.31	0.00	14.00
Wrt In Progress Fault Rate	0.00	0.00	0.00	0.00
System Fault Rate	0.00	0.00	0.00	0.00
Free List Size	99.00	94.19	72.00	111.00
Modified List Size	345.00	376.60	303.00	491.00

OpenVMS Monitor Utility
PAGE MANAGEMENT STATISTICS
on node ELLIS

5-SEP-1999 09:26:54.97

	CUR	AVE	MIN	MAX
Page Fault Rate	22863.55	23685.62	8012.97	31770.32
Page Read Rate	45.48	22.25	0.00	45.48
Page Read I/O Rate	23.36	10.28	0.00	23.36
Page Write Rate	1.24	11.00	0.00	21.33
Page Write I/O Rate	0.62	0.43	0.00	0.99
Free List Fault Rate	136.76	115.62	45.10	147.10
Modified List Fault Rate	22674.14	23525.52	7854.29	31632.25
Demand Zero Fault Rate	0.00	24.13	0.00	99.80
Global Valid Fault Rate	28.97	9.80	0.00	28.97
Wrt In Progress Fault Rate	0.00	0.00	0.00	0.00
System Fault Rate	0.00	0.00	0.00	0.00
Free List Size	140.00	101.83	72.00	140.00
Modified List Size	371.00	375.66	303.00	491.00

OpenVMS Monitor Utility
PAGE MANAGEMENT STATISTICS
on node ELLIS

5-SEP-1999 09:26:57.98

	CUR	AVE	MIN	MAX
Page Fault Rate	151.00	20710.37	151.00	31770.32
Page Read Rate	78.33	29.31	0.00	78.33
Page Read I/O Rate	35.00	13.39	0.00	35.00
Page Write Rate	0.00	9.61	0.00	21.33
Page Write I/O Rate	0.00	0.37	0.00	0.99
Free List Fault Rate	38.00	105.79	38.00	147.10
Modified List Fault Rate	57.66	20558.75	57.66	31632.25
Demand Zero Fault Rate	0.00	21.08	0.00	99.80
Global Valid Fault Rate	19.00	10.96	0.00	28.97
Wrt In Progress Fault Rate	0.00	0.00	0.00	0.00
System Fault Rate	0.00	0.00	0.00	0.00
Free List Size	433.00	149.14	72.00	433.00
Modified List Size	65.00	331.28	65.00	491.00

Working Sets and Automatic Working Set Adjustment

Module 5

Module Outline

Objectives

This module will describe the operations and characteristics of the working set and the working set list. Upon completion of this module, students should be able to calculate proper working set sizes for the WSDEFAULT, WSQUOTA, and WSEXTENT, which will prevent memory exhaustion.

Topics

- The Working Set List
- Working Set Size vs. Working Set List Size
- Parameters Affecting the Working Set List Size
- Working Set List Parameter Relationships
- Automatic Working Set Adjustment
- AWSA Mechanics
- Working Set Performance Considerations
- AWSA Performance Considerations

Working Set List

The need for the working set list:

- As the process generates faults and has them resolved, more pages become valid.
- As more pages become valid, fewer pages are available on the free page list.
- Some mechanism must limit the number of pages process may own to prevent the free list from being depleted. This mechanism is the working set list.

Working set list mechanics:

- Over time, size of list may vary due to automatic working set adjustment (**AWSA**) but at any given time, size of list is fixed.
- When list becomes full, that is, the number of pages in working set equals the size in entries of the list, a page must be displaced.
- Although more elaborate schemes could be used (Least Frequently Used or Least Recently Used), OpenVMS uses the fairly simplistic scheme of first-in-first-out (FIFO).
- This could be disastrous, if not for the fact that a page that is being heavily used may be soft faulted from the free or modified page list fairly quickly.

OpenVMS became more intelligent in page replacement scheme in version 3, using the **SYSGEN** parameter called **TBSKIPWSL**.

On some processors, this allows for not-recently-used (**NRU**) tendencies in page replacement.

It works in the following way:

- When we locate a page to displace from working set list, first check to see if the page is mapped in translation buffer (translation buffer is a limited size address translation cache; if page is mapped in translation buffer, it has been accessed fairly recently).
- If the page is mapped in translation buffer, skip displacing the page and examine the next entry. The process will continue until we have located a page that is not mapped in translation buffer or until we have checked **TBSKIPWSL** entries.

If we have checked **TBSKIPWSL** entries, the next page is displaced regardless of whether or not it is mapped in the translation buffer.

- Note that not all systems support this scheme.

Working Set Size vs. Working Set List Size

Working Set

- Set of pages valid for a process
- Pages are "owned" by process
- Pages are memory resident if process is a member of balance set
- Working set size of process is the number of pages "owned" by the process
- You may determine working set size of process by examining "physical memory" columns in **\$SHOW SYSTEM** and **\$SHOW STATUS** displays or using **F\$GETJPI** lexical function in the following way:

```
$WRITE SYS$OUTPUT F$GETJPI(pid, "PPGCNT")+F$GETJPI(pid, "GPGCNT")
```

 - **PPGCNT** is the number of process private pages in the working set.
 - **GPGCNT** is the number of global pages in the process' working set.

Working Set List

- Contiguous set of longwords identifying which pages are in the working set.
- The list limits the number of pages that may be valid.
- The working set list size must always be greater than or equal to the working set size.
- Defines maximum number of "owned" pages process may have valid before page replacement must occur.
- To determine the working set list size, examine the "working set" field in **\$SHOW STATUS** display or use **F\$GETJPI** lexical function in the following way:

```
$WRITE SYS$OUTPUT F$GETJPI(pid, "WSSIZE")
```

Parameters Affecting the Working Set List Size

Working Set Default or Limit

- Defines the size of the working set list at image activation or rundown.
- List size may change after initial setting due to automatic working set adjustment.
- The limit, or default, is set initially at process creation by the **WSDEFAULT** entry in **UAF** record. UAF entry affects processes that are created through **LOGINOUT** image (interactive login, batch process creation, network process creation and spawning). Default is defined through the **PQL_DWSDEFAULT** **SYSGEN** parameter, if the process is created through **\$RUN** DCL command or **SY\$CREPRC** system service.
- May be later adjusted up to working set quota, using **\$SET WORKING_SET/LIMIT=n**.

Working Set Quota

- Defines the size that the working set list may grow to under any circumstances.
- Should be viewed as memory commitment to processes on the system.
- Of all of the working set list parameters, the most caution should be taken in setting quotas.
- If quotas sum up to more memory than is available on the system, you risk exhausting memory and may experience a severe performance hit.
- Set by **WSQUOTA** entry in **UAF** record or the **PQL_DWSQUOTA** **SYSGEN** parameter, depending on how process was created (see default/limit).
- May lower quota or raise it to initial value using the **\$SET WORKING_SET/QUOTA=n** DCL command.

Working Set Extent

- Allows the process working set list to grow above its quota to its extent if plenty of free memory is available on the system (that is, if free list has **BORROWLIM (SYSGEN)** pages or more on it).
- Allows a process to use more memory when the system is lightly loaded.
- As load increases on the system, process will lose the ability to use these loans.
- You can minimize soft faults under varying workloads by setting larger extent.
- Extent is set, as before, by **WSEXTENT** entry in **UAF** or **PQL_DWSEXTENT SYSGEN** parameter and may be overridden by **\$SET WORKING_SET/EXTENT=*n***, up to authorized extent.



Note

The **SYSGEN** parameter **WSMAX** defines the absolute upper bound on the working set list size, over and the above any of above settings.

Automatic Working Set Adjustment

OpenVMS allows process working set lists to grow and shrink based on the memory requirements of process. This changing of the working set list size is called *Automatic Working Set Adjustment (AWSA)*.

- **AWSA** is implemented to allow for balancing of memory usage based on need and availability.
- Working set list (and not necessarily working set) is adjusted.
- Adjustment is performed automatically at the **QUANTUM** end, unless disabled for process or system.
- DCL command **\$SET WORK/NOADJUST** disables **AWSA** for process.
- Setting parameter **WSINC** to 0 will disable **AWSA** system-wide.

AWSA Mechanics

- **QUANTUM** is the CPU time interval in which process may use the processor before OpenVMS checks to determine whether it must give up the processor to another process of equal priority.

One operation performed at this interval is checking for **AWSA**.

- At the end of **QUANTUM**, OpenVMS checks to see whether process is also at the end of its adjustment interval.
 - The **AWSA** interval is defined by **SYSGEN** parameter **AWSTIME**.
 - If the interval has expired, the page fault rate of process is calculated, using the following formula:

$$\text{PFrate}/10\text{sec} = \text{page faults since last AWSTIME} * 1000 / \text{AWSTIME}$$

- Calculation yields the page fault rate per 10 seconds, even though page faults were sampled over interval **AWSTIME**, with units of 10 milliseconds (**AWSTIME** defaults to 20, taking value times 10ms yields a default sampling interval of 200ms or .2 sec.).
- The calculated page fault rate per 10 seconds is compared to setting for **SYSGEN** parameter **PFRATH**.
 - If the rate is higher than **PFRATH**, the process is paging enough to receive the increase of **WSINC** entries in its working set list.
 - If adjustment leaves the process with **WSQUOTA** entries or less, the increase will be granted.
 - If increase causes the list to grow above its quota, the increase will be granted, on the condition that **BORROWLIM** pages or more are on the free list.
 - If the increase should occur and
 - ♦ if Free Page List > **BORROWLIM**, the Working Set List size is set to $\text{MIN}(\text{Working Set List size} + \text{WSINC}, \text{WSEXTENT})$
 - ♦ else Working Set List size is set to $\text{MIN}(\text{Working Set List size} + \text{WSINC}, \text{WSQUOTA})$

- If the process is not paging at a rate greater than **PFRATH**, OpenVMS compares the page fault rate against **SYSGEN** parameter **PFRATL**.
 - If the rate is less than **PFRATL**, the system decreases the working set list size by **SYSGEN** parameter **WSDEC** entries.
 - Most likely, decreasing the list size will also cause the working set to decrease as well.
- If the paging rate is not high and **PFRATL** is set to 0, OpenVMS performs an additional check to determine whether a "dormant" process is tying up memory.
 - If the process is dormant, OpenVMS forces it back to its quota. The formal checks follow:

If **PFRATL = 0**, the free page list has less than **GROWLIM** pages, the process is dormant (received **PIXSCAN** boosts in last 32 **QUANTUM**), and the process working set list size is greater than its **WSQUOTA**, then the working set list size is set to the maximum of the process working set quota or its current working set list size minus **WSDEC**.
- The dormancy check is reason for a much higher setting for **WSDEC** in version 5+, because it allows a process that is dormant to be quickly forced back to its quota.

Working Set Performance Considerations

Setting WSDEFAULT

WSDEFAULT can be set to just about anything.

- OpenVMS myth-conception is that a smaller **WSDEFAULT** will limit memory usage when small applications are run.
- **WSDEFAULT** defines working set *list* size at image activation.
- P0 space (primary reason for working set size to grow) goes away at image rundown.
- A process running a small application will use small amount of memory regardless of **WSDEFAULT** setting.
- Setting the default smaller than **WSQUOTA** slightly slows activation of the image, as process must now wait for **AWSA** before the list can grow to quota.
- It is not unreasonable to set **WSDEFAULT** to equal **WSQUOTA**.

Setting WSQUOTA

Memory commitment to the process.

- Do NOT commit more memory than exists on the system.
- A simplistic but effective approach is to sum working set quotas for maximum workload and compare the sum to the amount of memory that is not OpenVMS memory.
- If the sum does not exceed the amount of memory that is not OpenVMS memory, the commitment is safe.
- Determine amount of memory that is not OpenVMS memory by issuing **\$SHOW MEMORY/PHYSICAL** and subtracting the amount of memory permanently allocated to OpenVMS and subtracting it from total memory on the system.
- It is probably safe to over-size quotas, if you can guarantee a large number of global pages shared by processes.

AWSA Performance Considerations

Sizing WSEXTENT

Should be used to allow processes to grow under lightly loaded configurations.

- Be safe with quotas and fairly liberal with extents.
- As extents increase, consider increasing **BORROWLIM** so that loans and quotas can co-exist peacefully.

BORROWLIM and GROWLIM

If installing images **/SHARE** to speed up image activation, consider increasing **BORROWLIM** and **GROWLIM** to prevent loans from exhausting the free page list.

- Experiment with these values—but **BORROWLIM** should be greater than **GROWLIM**.
- Set **BORROWLIM** to 1 to 3 times average **WSEXTENT**.
- This allows for growth on a memory-rich system, but preserves pages on the free list (might be useful for image activation).

Evaluating Working Sets Example

```

$
$
$ set process/privilege=cmkrnl
$
$ analyze/system

VAX/VMS System analyzer

SDA>
SDA> evaluate (@PFN$GL_PHYPGCNT)/(@PMSS$GL_PROCCNTMAX)
Hex = 00000285   Decimal = 645               BUG$_NOBUFCKT+00005
SDA>
SDA> evaluate @PMSS$GL_PROCCNTMAX
Hex = 00000013   Decimal = 19               EXE$V_EXPLICIT
                                           PR$_ASTLVL
                                           (remaining symbols suppressed by default)
SDA>
SDA> evaluate @PFN$GL_PHYPGCNT
Hex = 00002FE3   Decimal = 12259            UCB$M_TEMPLATE+00FE3
SDA>
SDA> exit

$ show memory/physical
      System Memory Resources on 24-OCT-1992 13:08:08.49
Physical Memory Usage (pages):  Total      Free    In Use  Modified
Main Memory (8.00Mb)          16384      3896   12400    88
Of the physical pages in use, 4125 pages are permanently allocated to VMS.
$
$ show work
Working Set      /Limit= 512   /Quota= 1024   /Extent= 2048
Adjustment enabled   Authorized Quota= 1024  Authorized Extent= 2048
$

```

Alpha AWSA Parameters

```
$
$
$ mcr sysgen
SYSGEN> SHOW/PQL
```

Parameters in use: Active

Parameter Name	Current	Default	Min.	Max.	Unit	Dynamic
-----	-----	-----	-----	-----	-----	-----
PQL_DASTLM	24	24	-1	-1	Ast	D
PQL_MASTLM	100	4	-1	-1	Ast	D
PQL_DBIOLM	32	32	-1	-1	I/O	D
PQL_MBIOLM	100	4	-1	-1	I/O	D
PQL_DBYTLM	65536	65536	-1	-1	Bytes	D
PQL_MBYTLM	100000	1024	-1	-1	Bytes	D
PQL_DCPULM	0	0	-1	-1	10Ms	D
PQL_MCPULM	0	0	-1	-1	10Ms	D
PQL_DDIOLM	32	32	-1	-1	I/O	D
PQL_MDIOLM	100	4	-1	-1	I/O	D
PQL_DFILLM	128	128	-1	-1	Files	D
PQL_MFILLM	100	2	-1	-1	Files	D
PQL_DPGFLQUOTA	65536	65536	-1	-1	Pagelets	D
internal value	4096	4096	0	-1	Pages	D
PQL_MPGFLQUOTA	32768	2048	-1	-1	Pagelets	D
internal value	2048	128	128	-1	Pages	D
PQL_DPRCLM	32	32	-1	-1	Processes	D
PQL_MPRCLM	10	0	-1	-1	Processes	D
PQL_DTQELM	16	16	-1	-1	Timers	D
PQL_MTQELM	0	0	-1	-1	Timers	D
PQL_DWSDEFAULT	1072	1024	-1	-1	Pagelets	
internal value	67	64	0	-1	Pages	
PQL_MWSDEFAULT	1600	512	-1	-1	Pagelets	
internal value	100	32	32	-1	Pages	
PQL_DWSQUOTA	2144	2048	-1	-1	Pagelets	D
internal value	134	128	0	-1	Pages	D
PQL_MWSQUOTA	2144	1024	-1	-1	Pagelets	D
internal value	134	64	64	-1	Pages	D
PQL_DWSEXTENT	16384	16384	-1	-1	Pagelets	D
internal value	1024	1024	0	-1	Pages	D
PQL_MWSEXTENT	16384	2048	-1	-1	Pagelets	D
internal value	1024	128	128	-1	Pages	D
...						

```

SYSGEN>
SYSGEN>  SHOW BORROWLIM
Parameter Name      Current      Default      Min.      Max.      Unit      Dynamic
-----
BORROWLIM           78          300          0          -1      Pages      D
SYSGEN>  SHOW GROWLIM
Parameter Name      Current      Default      Min.      Max.      Unit      Dynamic
-----
GROWLIM             78          63           0          -1      Pages      D
SYSGEN>  SHOW FREEGOAL
Parameter Name      Current      Default      Min.      Max.      Unit      Dynamic
-----
FREEGOAL            256         200          16          -1      Pages      D
SYSGEN>
SYSGEN>  SHOW WSDEC
Parameter Name      Current      Default      Min.      Max.      Unit      Dynamic
-----
WSDEC                4000         4000          0          -1      Pagelets    D
  internal value    250          250           0          -1      Pages      D
SYSGEN>  SHOW WSINC
Parameter Name      Current      Default      Min.      Max.      Unit      Dynamic
-----
WSINC                2400         2400          0          -1      Pagelets    D
  internal value    150          150           0          -1      Pages      D
SYSGEN>
SYSGEN>  SHOW MMG_CTL
Parameter Name      Current      Default      Min.      Max.      Unit      Dynamic
-----
MMG_CTLFLAGS         3            3            0          -1      Bit-mask    D
SYSGEN>  EXIT

```

\$

Sample Tuning Session

In the following scenario, we have a workstation with 32MB memory running V7.1 of OpenVMS. The analysis was based on the time taken to set up a new DECterm window. The first session used the standard **AUTOGEN** settings within the restrictions of DECwindows.

```
$ sh work
Working Set (pagelets) /Limit=1024 /Quota=2112 /Extent=16384
Adjustment enabled    Authorized Quota=2112  Authorized Extent=16384

Working Set (8Kb pages) /Limit=64 /Quota=132 /Extent=1024
                        Authorized Quota=132  Authorized Extent=1024

$ sh mem/phy
System Memory Resources on 6-SEP-1999 23:43:54.76

Physical Memory Usage (pages):    Total      Free      In Use    Modified
Main Memory (32.00Mb)            4096      107      3840      149

Of the physical pages in use, 2784 pages are permanently allocated to
OpenVMS.
$ def/key/term pf1 "show time"
%DCL-I-DEFKEY, DEFAULT key PF1 has been defined
$ def/key/term pf2 "monitor/display=test1.mon page,process"
%DCL-I-DEFKEY, DEFAULT key PF2 has been defined
$ show time
6-SEP-1999 23:47:24
$ monitor/display=test1.mon page,process
```

The window was requested at this point. The next SHOW TIME display was issued upon completion of the display of the window. It took 43 seconds to display the window.

```
$ show time
6-SEP-1999 23:48:07
$

$ sh proc

6-SEP-1999 23:49:55.69  User: SYSTEM      Process ID: 0000005D
                        Node: ELLIS      Process name: "_FTA1:"

Terminal:              FTA1:
User Identifier:       [SYSTEM]
Base priority:         4
Default file spec:     SYS$SYSR00T:[SYSMGR]
Number of Kthreads:    1

Devices allocated:     ELLIS$FTA1:
$
$ set def sys$system
$ r authorize
```

UAF> **show system**

```

Username: SYSTEM                      Owner:  SYSTEM MANAGER
Account:  SYSTEM                      UIC:    [1,4] ([SYSTEM])
CLI:      DCL                         Tables: DCLTABLES
Default:  SYS$SYSROOT:[SYSMGR]
LGICMD:
Flags:
Primary days:  Mon Tue Wed Thu Fri
Secondary days:                      Sat Sun
No access restrictions
Expiration:      (none)      Pwdminimum:  8      Login Fails:      0
Pwdlifetime:     999 00:00    Pwdchange:   7-JUL-1999 17:15
Last Login:  6-SEP-1999 23:42 (interactive), 6-SEP-1999 23:47
(non-interactive)
Maxjobs:         0  Fillm:      100  Bytlm:      64000
Maxacctjobs:     0  Shrfillm:    0  Pbytlm:      0
Maxdetach:       0  BIOlm:     150  JTquota:    4096

```

Note

The requested working set sizes differ from those actually in use. This is due to the PQL_Mxxxx settings.

```

Prclm:          10  DI0lm:          150  WSdef:          1024
Prio:            4  ASTlm:          250  WSquo:          1024
Queprio:         0  TQEIm:          200  WSextent:        2048
CPU:             (none)  Enqlm:        2000  Pgflquo:        50000
Authorized Privileges:
  ACNT      ALLSP00L    ALTPRI      AUDIT      BUGCHK      BYPASS
  CMEXEC    CMKRNL     DIAGNOSE  DOWNGRADE  EXQUOTA     GROUP
  GRPNAM    GRPPRV     IMPERSONATE  IMPORT     LOG_IO      MOUNT
  NETMBX    OPER       PFNMAP      PHY_IO     PRMCEB      PRMGBL
  PRMMBX    PSWAPM     READALL     SECURITY   SETPRV      SHARE
  SHMEM     SYSGBL     SYSLCK      SYSNAM     SYSPRV      TMPMBX
  UPGRADE   VOLPRO     WORLD
Default Privileges:
  ACNT      ALLSP00L    ALTPRI      AUDIT      BUGCHK      BYPASS
  CMEXEC    CMKRNL     DIAGNOSE  DOWNGRADE  EXQUOTA     GROUP
  GRPNAM    GRPPRV     IMPERSONATE  IMPORT     LOG_IO      MOUNT
  NETMBX    OPER       PFNMAP      PHY_IO     PRMCEB      PRMGBL
  PRMMBX    PSWAPM     READALL     SECURITY   SETPRV      SHARE
  SHMEM     SYSGBL     SYSLCK      SYSNAM     SYSPRV      TMPMBX
  UPGRADE   VOLPRO     WORLD
Identifier      Value      Attributes
  NET$MANAGE     %X91F50002
UAF> ex
$ mcr sysgen
SYSGEN> SET/OUT=TEST1.SYSGEN
SYSGEN> SHOW/ALL
SYSGEN> SHOW/SPECIAL
SYSGEN> Exit

```

\$

\$ TYPE test1.mon
Process Count: 22

OpenVMS Monitor Utility
PROCESSES

Uptime: 0 00:09:47

on node ELLIS

6-SEP-1999 23:47:32.31

PID	STATE	PRI	NAME	PAGES	DIOCNT	FAULTS	CPU TIME
00000041	HIB	16	SWAPPER	0/0	0	0 00:00:01.9	
00000044	HIBO	13	LANACP	1/28		SWAPPED OUT	
00000046	HIBO	10	IPCACP	0/20		SWAPPED OUT	
00000047	HIBO	8	ERRFMT	0/29		SWAPPED OUT	
00000048	HIBO	7	OPCOM	0/25		SWAPPED OUT	
00000049	HIB	9	AUDIT_SERVER	2/90	29	190 00:00:00.2	
0000004A	HIBO	9	JOB_CONTROL	0/35		SWAPPED OUT	
0000004B	HIBO	9	QUEUE_MANAGER	0/32		SWAPPED OUT	
0000004C	HIB	10	SECURITY_SERVER	8/91	23	885 00:00:01.8	
0000004D	LEFO	5	DNS\$ADVER	0/31		SWAPPED OUT	
0000004E	HIBO	8	LESSACP_V30	0/32		SWAPPED OUT	
0000004F	HIBO	8	REMACP	0/16		SWAPPED OUT	
00000050	HIBO	4	NET\$ACP	2/27		SWAPPED OUT	
00000051	HIBO	4	NET\$EVD	0/26		SWAPPED OUT	
00000052	LEFO	10	DTSS\$CLERK	0/32		SWAPPED OUT	
00000053	HIBO	8	TP_SERVER	1/40		SWAPPED OUT	
00000055	HIB	6	DECW\$SERVER_0	18/375	629	2667 00:00:09.8	
00000056	COMO	4	DTSESSION	0/27		SWAPPED OUT	
0000005A	PFW	4	DTWM	59/158	457	12243 00:00:16.0	
0000005C	LEF	5	DECW\$TE_005C	13/35	115	9751 00:00:09.9	
0000005D	CUR	5	_FTA1:	5/35	37	677 00:00:00.7	
00000064	PFW	7	SYSTEM\$CDE002	0/31	0	26 00:00:00.0	

OpenVMS Monitor Utility
PAGE MANAGEMENT STATISTICS
on node ELLIS
6-SEP-1999 23:47:35.93



Note

Notice the Hard fault rate.

	CUR	AVE	MIN	MAX
Page Fault Rate	245.70	175.07	90.66	245.70
Page Read Rate	114.12	82.02	43.66	114.12
Page Read I/O Rate	45.70	35.80	24.00	45.70
Page Write Rate	0.00	9.66	0.00	21.33
Page Write I/O Rate	0.00	0.15	0.00	0.33
Free List Fault Rate	56.23	41.08	23.00	56.23
Modified List Fault Rate	108.03	73.71	32.66	108.03
Demand Zero Fault Rate	5.26	5.89	5.26	6.66
Global Valid Fault Rate	29.36	17.97	4.33	29.36
Wrt In Progress Fault Rate	0.00	0.00	0.00	0.00
System Fault Rate	0.00	0.15	0.00	0.33
Free List Size	89.00	151.50	89.00	214.00
Modified List Size	282.00	254.50	227.00	282.00

Process Count: 22

OpenVMS Monitor Utility
 PROCESSES
 on node ELLIS
 6-SEP-1999 23:47:39.14

Uptime: 0 00:09:54

PID	STATE	PRI	NAME	PAGES	DIOCNT	FAULTS	CPU TIME
00000041	HIB	16	SWAPPER	0/0	0	0	00:00:01.9
00000044	HIBO	13	LANACP	1/28		SWAPPED OUT	
00000046	HIBO	10	IPCACP	0/20		SWAPPED OUT	
00000047	HIBO	8	ERRFMT	0/29		SWAPPED OUT	
00000048	HIBO	7	OPCOM	0/25		SWAPPED OUT	
00000049	HIB	9	AUDIT_SERVER	2/90	29	190	00:00:00.2
0000004A	HIBO	9	JOB_CONTROL	0/35		SWAPPED OUT	
0000004B	HIBO	9	QUEUE_MANAGER	0/32		SWAPPED OUT	
0000004C	HIB	10	SECURITY_SERVER	8/94	23	892	00:00:01.9
0000004D	LEFO	5	DNS\$ADVER	0/31		SWAPPED OUT	
0000004E	HIBO	8	LES\$ACP_V30	0/32		SWAPPED OUT	
0000004F	HIBO	8	REMACP	0/16		SWAPPED OUT	
00000050	HIB	4	NET\$ACP	9/33	34	798	00:00:00.5
00000051	HIBO	4	NET\$EVD	0/26		SWAPPED OUT	
00000052	LEFO	10	DTSS\$CLERK	0/32		SWAPPED OUT	
00000053	HIBO	8	TP_SERVER	0/29		SWAPPED OUT	
00000055	HIB	8	DECW\$SERVER_0	18/375	633	2667	00:00:09.8
00000056	LEF	4	DTSESSION	5/27	290	3441	00:00:03.8
0000005A	PFW	4	DTWM	12/35	469	12427	00:00:16.2
0000005C	LEF	4	DECW\$TE_005C	9/35	115	9929	00:00:09.9
0000005D	CUR	4	_FTA1:	6/36	38	1108	00:00:00.8
00000064	PFW	4	SYSTEM	46/132	29	243	00:00:00.6

OpenVMS Monitor Utility
 PAGE MANAGEMENT STATISTICS
 on node ELLIS
 6-SEP-1999 23:47:42.31

	CUR	AVE	MIN	MAX
Page Fault Rate	378.48	210.91	90.66	378.48
Page Read Rate	95.88	85.54	43.66	114.12
Page Read I/O Rate	43.98	38.27	24.00	45.70
Page Write Rate	7.27	21.44	0.00	59.81
Page Write I/O Rate	0.94	0.53	0.00	0.94
Free List Fault Rate	91.77	48.11	19.93	91.77
Modified List Fault Rate	171.83	88.39	32.66	171.83
Demand Zero Fault Rate	15.50	8.99	5.26	15.50
Global Valid Fault Rate	54.43	24.67	4.33	54.43
Wrt In Progress Fault Rate	0.00	1.22	0.00	4.98
System Fault Rate	0.00	0.07	0.00	0.33
Free List Size	101.00	134.00	89.00	214.00
Modified List Size	289.00	266.00	227.00	289.00

Process Count: 22 OpenVMS Monitor Utility Uptime: 0 00:10:00
PROCESSES
on node ELLIS
6-SEP-1999 23:47:45.48

PID	STATE	PRI	NAME	PAGES	DIOCNT	FAULTS	CPU TIME
00000041	HIB	16	SWAPPER	0/0	0	0	00:00:02.0
00000044	HIBO	13	LANACP	1/28		SWAPPED OUT	
00000046	HIB	10	IPCACP	0/20	3	32	00:00:00.0
00000047	HIBO	8	ERRFMT	0/29		SWAPPED OUT	
00000048	HIBO	7	OPCOM	0/25		SWAPPED OUT	
00000049	HIB	9	AUDIT_SERVER	2/90	29	190	00:00:00.2
0000004A	HIBO	9	JOB_CONTROL	0/35		SWAPPED OUT	
0000004B	HIBO	9	QUEUE_MANAGER	0/32		SWAPPED OUT	
0000004C	HIB	10	SECURITY_SERVER	8/97	23	894	00:00:01.9
0000004D	LEFO	5	DNS\$ADVER	0/31		SWAPPED OUT	
0000004E	HIBO	8	LESSACP_V30	0/32		SWAPPED OUT	
0000004F	HIBO	8	REMACP	0/16		SWAPPED OUT	
00000050	HIBO	4	NET\$ACP	0/24		SWAPPED OUT	
00000051	HIBO	4	NET\$EVD	0/26		SWAPPED OUT	
00000052	LEFO	10	DTSS\$CLERK	0/32		SWAPPED OUT	
00000053	HIBO	8	TP_SERVER	0/29		SWAPPED OUT	
00000055	HIB	6	DECW\$SERVER_0	28/375	633	2687	00:00:09.9
00000056	LEFO	4	DTSESSION	0/22		SWAPPED OUT	
0000005A	LEF	4	DTWM	39/132	470	13073	00:00:16.4
0000005C	PFW	4	DECW\$TE_005C	60/127	115	10176	00:00:10.0
0000005D	CUR	4	_FTA1:	5/35	38	1589	00:00:00.8
00000064	LEF	4	SYSTEM	8/35	65	864	00:00:01.1

OpenVMS Monitor Utility
PAGE MANAGEMENT STATISTICS
on node ELLIS
6-SEP-1999 23:47:48.62

	CUR	AVE	MIN	MAX
Page Fault Rate	229.16	221.64	90.66	378.48
Page Read Rate	137.17	97.25	43.66	137.17
Page Read I/O Rate	62.50	43.65	24.00	62.50
Page Write Rate	0.00	24.96	0.00	64.03
Page Write I/O Rate	0.00	0.56	0.00	1.26
Free List Fault Rate	55.44	51.68	19.93	91.77
Modified List Fault Rate	87.82	90.47	32.66	171.83
Demand Zero Fault Rate	0.00	6.05	0.00	15.50
Global Valid Fault Rate	23.07	27.96	4.33	54.43
Wrt In Progress Fault Rate	0.00	0.82	0.00	4.98
System Fault Rate	0.32	0.10	0.00	0.33
Free List Size	221.00	169.66	89.00	261.00
Modified List Size	255.00	241.16	128.00	289.00

Process Count: 22

OpenVMS Monitor Utility
 PROCESSES
 on node ELLIS
 6-SEP-1999 23:47:52.03

Uptime: 0 00:10:07

PID	STATE	PRI	NAME	PAGES	DIOCNT	FAULTS	CPU TIME
00000041	HIB	16	SWAPPER	0/0	0	0	00:00:02.0
00000044	HIBO	13	LANACP	1/28		SWAPPED OUT	
00000046	HIBO	10	IPCACP	0/20		SWAPPED OUT	
00000047	HIBO	8	ERRFMT	0/29		SWAPPED OUT	
00000048	HIBO	7	OPCOM	0/25		SWAPPED OUT	
00000049	HIB	9	AUDIT_SERVER	2/90	29	190	00:00:00.2
0000004A	HIBO	9	JOB_CONTROL	0/35		SWAPPED OUT	
0000004B	HIBO	9	QUEUE_MANAGER	0/32		SWAPPED OUT	
0000004C	HIB	10	SECURITY_SERVER	8/97	23	894	00:00:01.9
0000004D	LEFO	5	DNS\$ADVER	0/31		SWAPPED OUT	
0000004E	HIBO	8	LES\$ACP_V30	0/32		SWAPPED OUT	
0000004F	HIBO	8	REMACP	0/16		SWAPPED OUT	
00000050	HIBO	4	NET\$ACP	0/24		SWAPPED OUT	
00000051	HIBO	4	NET\$EVD	0/26		SWAPPED OUT	
00000052	LEFO	10	DTSS\$CLERK	0/32		SWAPPED OUT	
00000053	HIBO	8	TP_SERVER	0/29		SWAPPED OUT	
00000055	HIB	6	DECW\$SERVER_0	28/369	645	2820	00:00:10.0
00000056	LEFO	4	DTSESSION	0/22		SWAPPED OUT	
0000005A	PFW	4	DTWM	16/92	470	13124	00:00:16.4
0000005C	LEF	4	DECW\$TE_005C	64/132	115	11071	00:00:10.6
0000005D	CUR	4	_FTA1:	6/36	39	1952	00:00:00.9
00000064	PFW	4	SYSTEM	13/56	65	948	00:00:01.1

OpenVMS Monitor Utility
 PAGE MANAGEMENT STATISTICS
 on node ELLIS
 6-SEP-1999 23:47:56.00

	CUR	AVE	MIN	MAX
Page Fault Rate	319.14	238.16	90.66	378.48
Page Read Rate	125.44	104.23	43.66	137.17
Page Read I/O Rate	52.14	46.36	24.00	62.50
Page Write Rate	1.00	20.59	0.00	64.03
Page Write I/O Rate	0.25	0.48	0.00	1.26
Free List Fault Rate	73.29	55.65	19.93	91.77
Modified List Fault Rate	143.32	98.72	32.66	171.83
Demand Zero Fault Rate	9.57	5.80	0.00	15.50
Global Valid Fault Rate	39.29	30.03	4.33	54.43
Wrt In Progress Fault Rate	0.00	0.59	0.00	4.98
System Fault Rate	0.00	0.07	0.00	0.33
Free List Size	83.00	162.62	83.00	261.00
Modified List Size	232.00	238.37	128.00	289.00

Process Count: 22

OpenVMS Monitor Utility

Uptime: 0 00:10:18

PROCESSES
on node ELLIS
6-SEP-1999 23:48:03.24

PID	STATE	PRI	NAME	PAGES	DIOCNT	FAULTS	CPU TIME
00000041	HIB	16	SWAPPER	0/0	0	0	00:00:02.1
00000044	HIBO	13	LANACP	1/28		SWAPPED OUT	
00000046	HIBO	10	IPCACP	0/20		SWAPPED OUT	
00000047	HIBO	8	ERRFMT	0/29		SWAPPED OUT	
00000048	HIBO	7	OPCOM	0/25		SWAPPED OUT	
00000049	HIB	9	AUDIT_SERVER	2/90	29	190	00:00:00.2
0000004A	HIBO	8	JOB_CONTROL	0/27		SWAPPED OUT	
0000004B	HIBO	9	QUEUE_MANAGER	0/32		SWAPPED OUT	
0000004C	HIB	10	SECURITY_SERVER	8/86	23	907	00:00:01.9
0000004D	LEFO	5	DNS\$ADVER	0/31		SWAPPED OUT	
0000004E	HIBO	8	LESSACP_V30	0/32		SWAPPED OUT	
0000004F	HIBO	8	REMACP	0/16		SWAPPED OUT	
00000050	HIBO	4	NET\$ACP	0/24		SWAPPED OUT	
00000051	HIBO	4	NET\$EVD	0/26		SWAPPED OUT	
00000052	LEFO	10	DTSS\$CLERK	0/32		SWAPPED OUT	
00000053	HIBO	8	TP_SERVER	1/38		SWAPPED OUT	
00000055	HIB	8	DECW\$SERVER_0	28/369	682	2842	00:00:10.2
00000056	LEF	4	DTSESSION	36/99	293	3585	00:00:03.9
0000005A	LEF	4	DTWM	66/216	470	13478	00:00:16.5
0000005C	COMO	4	DECW\$TE_005C	23/97		SWAPPED OUT	
0000005D	CUR	6	_FTA1:	53/132	39	2344	00:00:01.1
00000065	LEFO	4	_FTA5:	17/34		SWAPPED OUT	

OpenVMS Monitor Utility
PAGE MANAGEMENT STATISTICS
on node ELLIS
6-SEP-1999 23:48:06.25

	CUR	AVE	MIN	MAX
Page Fault Rate	38.87	192.63	19.73	378.48
Page Read Rate	16.94	88.01	8.88	137.17
Page Read I/O Rate	7.30	39.24	5.59	62.50
Page Write Rate	40.19	19.13	0.00	64.03
Page Write I/O Rate	0.99	0.51	0.00	1.26
Free List Fault Rate	14.28	44.79	4.27	91.77
Modified List Fault Rate	12.95	78.32	4.60	171.83
Demand Zero Fault Rate	0.00	4.19	0.00	15.50
Global Valid Fault Rate	3.65	24.16	1.97	54.43
Wrt In Progress Fault Rate	0.66	1.10	0.00	4.98
System Fault Rate	0.00	0.05	0.00	0.33
Free List Size	186.00	147.90	64.00	261.00
Modified List Size	184.00	228.09	128.00	289.00

\$

The following session worked with smaller PQL_Mxxxx settings and larger BORROWLIM and FREEGOAL settings. Notice the smaller working set sizes.

```
$ sh work
Working Set (pagelets) /Limit=1024 /Quota=1024 /Extent=4096
Adjustment enabled      Authorized Quota=1024  Authorized Extent=4096

Working Set (8Kb pages) /Limit=64 /Quota=64 /Extent=256
                        Authorized Quota=64  Authorized Extent=256

$ show mem/phy
      System Memory Resources on  7-SEP-1999 00:08:23.23

Physical Memory Usage (pages):      Total      Free      In Use      Modified
Main Memory (32.00Mb)              4096        43      3540        513

Of the physical pages in use, 2733 pages are permanently allocated to OpenVMS.
$ def/key/term pf1 "show time"
%DCL-I-DEFKEY, DEFAULT key PF1 has been defined
$ def/key/term pf2 "monitor/display=test2.mon page,process"
%DCL-I-DEFKEY, DEFAULT key PF2 has been defined
$
$ show time
      7-SEP-1999 00:09:44
$ monitor/display=test2.mon page,process
```

Once again, window was requested at this point and completed on issue of the following \$SHOW TIME command. This window took only 38 seconds to complete.

```
$ show time
      7-SEP-1999 00:10:22
$ mcr sysgen
SYSGEN> SET/OUT=TEST2.SYSGEN
SYSGEN> SHOW/ALL
SYSGEN> SHOW/SPECIAL
SYSGEN> EXIT
$
```

The following lists the differences between the SYSGEN parameter settings.

\$ diff test1.sysgen test2.sysgen

File SYS\$SYSROOT:[SYSMGR]TEST1.SYSGEN;1

6	GBLPAGES	176184	30720	10240	-1
Pagele					
7	internal value	11012	1920	640	-1 Pages
8	GBLPAGFIL	1024	128	32	-1 Pages
9	MAXPROCESSCNT	49	32	12	16384
Proces					
10	MULTITHREAD	1	1	0	16 KThrea

File SYS\$SYSROOT:[SYSMGR]TEST2.SYSGEN;1

6	GBLPAGES	176141	30720	10240	-1 Pagele
7	internal value	11009	1920	640	-1 Pages
8	GBLPAGFIL	1024	128	32	-1 Pages
9	MAXPROCESSCNT	35	32	12	16384
Proces					
10	MULTITHREAD	1	1	0	16 KThrea

File SYS\$SYSROOT:[SYSMGR]TEST1.SYSGEN;1

21	SYSMWCNT	2933	2048	512	1048576 Pagele
22	internal value	184	128	32	65536 Pages
23	KSTACKPAGES	2	1	1	768 Pages
24	BALSETCNT	47	30	8	16384 Slots
25	WSMAX	16384	4096	1024	8388608 Pagele
26	internal value	1024	256	64	524288 Pages
27	NPAGEDYN	1646592	1048576	163840	-1 Bytes
28	NPAGEVIR	8232960	8388608	163840	-1 Bytes
29	PAGEDYN	737280	212992	65536	-1 Bytes
30	QUANTUM	20	20	2	32767 10Ms

File SYS\$SYSROOT:[SYSMGR]TEST2.SYSGEN;1

21	SYSMWCNT	2783	2048	512	1048576 Pagele
22	internal value	174	128	32	65536 Pages
23	KSTACKPAGES	2	1	1	768 Pages
24	BALSETCNT	33	30	8	16384 Slots
25	WSMAX	16384	4096	1024	8388608 Pagele
26	internal value	1024	256	64	524288 Pages
27	NPAGEDYN	1474560	1048576	163840	-1 Bytes
28	NPAGEVIR	7372800	8388608	163840	-1 Bytes
29	PAGEDYN	622592	212992	65536	-1 Bytes
30	QUANTUM	20	20	2	32767 10Ms

File SYS\$SYSR00T:[SYSMGR]TEST1.SYSGEN;1

33	MPW_LOLIMIT	141	16	0	-1 Pages
34	MPW_IOLIMIT	4	4	1	127 I/O
35	MPW_THRESH	282	16	0	-1 Pages
36	MPW_WAITLIMIT	1280	576	64	-1 Pages

File SYS\$SYSR00T:[SYSMGR]TEST2.SYSGEN;1

33	MPW_LOLIMIT	99	16	0	-1 Pages
34	MPW_IOLIMIT	4	4	1	127 I/O
35	MPW_THRESH	198	16	0	-1 Pages
36	MPW_WAITLIMIT	1280	576	64	-1 Pages

**Note**

Notice the differences between FREEGOAL, GROWLIM, and BORROWLIM.

File SYS\$SYSR00T:[SYSMGR]TEST1.SYSGEN;1

67	FREELIM	67	32	16	-1 Pages
68	FREEGOAL	256	200	16	-1 Pages
69	GROWLIM	67	63	0	-1 Pages
70	BORROWLIM	67	300	0	-1 Pages
71	XFMAXRATE	236	236	0	255 Specia

File SYS\$SYSR00T:[SYSMGR]TEST2.SYSGEN;1

67	FREELIM	53	32	16	-1 Pages
68	FREEGOAL	700	200	16	-1 Pages
69	GROWLIM	800	63	0	-1 Pages
70	BORROWLIM	1000	300	0	-1 Pages
71	XFMAXRATE	236	236	0	255 Specia

**Note**

Notice the differences between the PQL_Mxxxxxx parameters

File SYS\$SYSR00T:[SYSMGR]TEST1.SYSGEN;1

171	PQL_DWSQUOTA	2112	2048	-1	-1 Pagele
172	internal value	132	128	0	-1 Pages
173	PQL_MWSQUOTA	2112	1024	-1	-1 Pagele
174	internal value	132	64	64	-1 Pages
175	PQL_DWSEXTENT	16384	16384	-1	-1 Pagele
176	internal value	1024	1024	0	-1 Pages
177	PQL_MWSEXTENT	16384	2048	-1	-1 Pagele
178	internal value	1024	128	128	-1 Pages
179	PQL_DENQLM	128	128	-1	-1 Locks

```

File SYS$SYSR00T:[SYSMGR]TEST2.SYSGEN;1
 171 PQL_DWSQUOTA          1024      2048      -1      -1 Pagele
 172 internal value         64        128        0      -1 Pages
 173 PQL_MWSQUOTA          1024      1024      -1      -1 Pagele
 174 internal value         64         64        64      -1 Pages
 175 PQL_DWSEXTENT         4096     16384      -1      -1 Pagele
 176 internal value        256       1024        0      -1 Pages
 177 PQL_MWSEXTENT         4096      2048      -1      -1 Pagele
 178 internal value        256        128      128      -1 Pages
 179 PQL_DENQLM            128        128      -1      -1 Locks
*****
File SYS$SYSR00T:[SYSMGR]TEST1.SYSGEN;1
 183 ACP_MAPCACHE          23         9         2      -1 Blocks
 184 ACP_HDRCACHE          94        36         8      -1 Blocks
 185 ACP_DIRCACHE          94        22         4      -1 Blocks
 186 ACP_DINDXCACHE        23        26         2      -1 Blocks
 187 ACP_WORKSET            0         0         0      -1 Pagele
*****
File SYS$SYSR00T:[SYSMGR]TEST2.SYSGEN;1
 183 ACP_MAPCACHE          16         9         2      -1 Blocks
 184 ACP_HDRCACHE          66        36         8      -1 Blocks
 185 ACP_DIRCACHE          66        22         4      -1 Blocks
 186 ACP_DINDXCACHE        16        26         2      -1 Blocks
 187 ACP_WORKSET            0         0         0      -1 Pagele
*****
*****
File SYS$SYSR00T:[SYSMGR]TEST1.SYSGEN;1
 192 ACP_SYSACC            11         8         0      -1 Direct
 193 ACP_MAXREAD           32        32         1      64 Blocks
*****
File SYS$SYSR00T:[SYSMGR]TEST2.SYSGEN;1
 192 ACP_SYSACC            8         8         0      -1 Direct
 193 ACP_MAXREAD           32        32         1      64 Blocks
*****
*****
File SYS$SYSR00T:[SYSMGR]TEST1.SYSGEN;1
 261 GH_RSRVPGCNT          20         0         0      2048 Pages
 262 NISCS_MAX_PKT SZ     1498     1498     1080     8192 Bytes
*****
File SYS$SYSR00T:[SYSMGR]TEST2.SYSGEN;1
 261 GH_RSRVPGCNT          0         0         0      2048 Pages
 262 NISCS_MAX_PKT SZ     1498     1498     1080     8192 Bytes
*****
*****
File SYS$SYSR00T:[SYSMGR]TEST1.SYSGEN;1
 282 NPAG_BAP_MAX_PA        0        -1         0      -1 PhysAd
 283 NPAG_RING_SIZE       2048     2048         0      -1 Entries
*****
File SYS$SYSR00T:[SYSMGR]TEST2.SYSGEN;1
 282 NPAG_BAP_MAX_PA       -1        -1         0      -1 PhysAd
 283 NPAG_RING_SIZE       2048     2048         0      -1 Entries
*****
*****

```

```

File SYS$SYSROOT:[SYSMGR]TEST1.SYSGEN;1
  291  PIXSCAN                4          1          0      8192 Proces
  292  SMP_CPUSH              0          0          0      -1 CPU bi
*****
File SYS$SYSROOT:[SYSMGR]TEST2.SYSGEN;1
  291  PIXSCAN                3          1          0      8192 Proces
  292  SMP_CPUSH              0          0          0      -1 CPU bi
*****
File SYS$SYSROOT:[SYSMGR]TEST1.SYSGEN;1
  323  LOCKIDTBL_MAX          42880     16776959      1792     16776959 Obsole
  324  PE1                    0          0          0          -1
*****
File SYS$SYSROOT:[SYSMGR]TEST2.SYSGEN;1
  323  LOCKIDTBL_MAX          38400     16776959      1792     16776959 Obsole
  324  PE1                    0          0          0          -1
*****
Number of difference sections found: 11
Number of difference records found: 37

```

```

DIFFERENCES /IGNORE=()/MERGED=1-
  SYS$SYSROOT:[SYSMGR]TEST1.SYSGEN;1-
  SYS$SYSROOT:[SYSMGR]TEST2.SYSGEN;1

```

\$

\$ type test2.mon/pag

```

Process Count: 21          OpenVMS Monitor Utility          Uptime:    0
00:07:19

```

PROCESSES

on node ELLIS

7-SEP-1999 00:09:53.28

PID	STATE	PRI	NAME	PAGES	DIOCNT	FAULTS	CPU TIME
00000041	HIB	16	SWAPPER	0/0		0	0 00:00:01.6
00000044	HIBO	13	LANACP	1/29		SWAPPED OUT	
00000046	HIBO	10	IPCACP	0/20		SWAPPED OUT	
00000047	HIBO	9	ERRFMT	0/30		SWAPPED OUT	
00000048	HIBO	7	OPCOM	0/27		SWAPPED OUT	
00000049	HIB	10	AUDIT_SERVER	1/43	33	265	00:00:00.2
0000004A	HIBO	9	JOB_CONTROL	0/24		SWAPPED OUT	
0000004B	HIBO	9	QUEUE_MANAGER	0/33		SWAPPED OUT	
0000004C	HIB	10	SECURITY_SERVER	8/97	25	1064	00:00:01.5
0000004D	LEFO	5	DNS\$ADVER	0/37		SWAPPED OUT	
0000004E	HIBO	8	LES\$ACP_V30	0/32		SWAPPED OUT	
0000004F	HIBO	8	REMACP	0/16		SWAPPED OUT	
00000050	HIBO	4	NET\$ACP	0/25		SWAPPED OUT	
00000051	HIBO	4	NET\$EVD	0/29		SWAPPED OUT	
00000052	LEFO	11	DTSS\$CLERK	0/30		SWAPPED OUT	
00000053	HIBO	8	TP_SERVER	0/40		SWAPPED OUT	
00000055	HIB	8	DECW\$SERVER_0	18/299	433	1631	00:00:07.4
00000056	LEFO	4	DTSESSION	0/30		SWAPPED OUT	
0000005A	PFW	4	DTWM	14/64	453	127389	00:00:19.3
0000005C	LEF	5	DECW\$TE_005C	12/35	116	62493	00:00:10.1
0000005D	CUR	4	_FTA1:	5/34	38	1315	00:00:00.8

Note

Notice the increase in soft faults.

OpenVMS Monitor Utility
PAGE MANAGEMENT STATISTICS
on node ELLIS

7-SEP-1999 00:09:56.34

	CUR	AVE	MIN	MAX
Page Fault Rate	1706.55	1135.13	563.69	1706.55
Page Read Rate	94.75	122.16	94.75	150.16
Page Read I/O Rate	38.36	48.76	38.36	59.40
Page Write Rate	0.00	0.49	0.00	0.99
Page Write I/O Rate	0.00	0.16	0.00	0.33
Free List Fault Rate	516.06	331.52	146.86	516.06
Modified List Fault Rate	970.49	639.57	308.58	970.49
Demand Zero Fault Rate	0.32	0.16	0.00	0.32
Global Valid Fault Rate	180.65	114.12	47.52	180.65
Wrt In Progress Fault Rate	0.00	0.00	0.00	0.00
System Fault Rate	0.00	0.00	0.00	0.00
Free List Size	67.00	73.50	67.00	80.00
Modified List Size	648.00	658.50	648.00	669.00

Process Count: 22

OpenVMS Monitor Utility

Uptime: 0 00:07:25

PROCESSES

on node ELLIS

7-SEP-1999 00:09:59.46

PID	STATE	PRI	NAME	PAGES	DIOCNT	FAULTS	CPU TIME
00000041	HIB	16	SWAPPER	0/0	0	0	00:00:01.7
00000044	HIBO	13	LANACP	1/29		SWAPPED OUT	
00000046	COMO	11	IPCACP	0/20		SWAPPED OUT	
00000047	HIBO	9	ERRFMT	0/30		SWAPPED OUT	
00000048	HIBO	7	OPCOM	0/27		SWAPPED OUT	
00000049	HIB	10	AUDIT_SERVER	1/43	33	265	00:00:00.2
0000004A	HIBO	9	JOB_CONTROL	0/24		SWAPPED OUT	
0000004B	HIBO	9	QUEUE_MANAGER	0/33		SWAPPED OUT	
0000004C	HIB	10	SECURITY_SERVER	8/97	25	1064	00:00:01.5
0000004D	LEFO	5	DNS\$ADVER	0/37		SWAPPED OUT	
0000004E	HIBO	8	LESSACP_V30	0/32		SWAPPED OUT	
0000004F	HIBO	8	REMACP	0/16		SWAPPED OUT	
00000050	HIBO	4	NET\$ACP	0/25		SWAPPED OUT	
00000051	HIBO	4	NET\$EVD	0/29		SWAPPED OUT	
00000052	LEFO	11	DTSS\$CLERK	0/30		SWAPPED OUT	
00000053	COMO	11	TP_SERVER	0/40		SWAPPED OUT	
00000055	HIB	8	DECW\$SERVER_0	18/310	435	1638	00:00:07.5
00000056	COMO	4	DTSESSION	0/30		SWAPPED OUT	
0000005A	PFW	4	DTWM	12/64	457	133189	00:00:19.9
0000005C	LEFO	5	DECW\$TE_005C	5/28		SWAPPED OUT	
0000005D	CUR	4	_FTA1:	5/34	39	1678	00:00:00.9
0000005E	PFW	4	SYSTEM\$CDE001	4/46	0	31	00:00:00.0

OpenVMS Monitor Utility
 PAGE MANAGEMENT STATISTICS
 on node ELLIS
 7-SEP-1999 00:10:02.50

	CUR	AVE	MIN	MAX
Page Fault Rate	880.92	864.24	319.23	1706.55
Page Read Rate	129.27	116.00	91.02	150.16
Page Read I/O Rate	53.61	46.77	36.21	59.40
Page Write Rate	0.00	24.81	0.00	96.47
Page Write I/O Rate	0.00	0.57	0.00	1.92
Free List Fault Rate	260.19	251.83	88.14	516.06
Modified List Fault Rate	493.75	483.02	166.98	970.49
Demand Zero Fault Rate	6.90	3.51	0.00	6.90
Global Valid Fault Rate	65.78	78.44	21.15	180.65
Wrt In Progress Fault Rate	0.00	0.00	0.00	0.00
System Fault Rate	0.00	0.00	0.00	0.00
Free List Size	191.00	159.25	67.00	299.00
Modified List Size	395.00	522.50	378.00	669.00

Process Count: 22 OpenVMS Monitor Utility Uptime: 0 00:07:31
 PROCESSES
 on node ELLIS
 7-SEP-1999 00:10:05.70

PID	STATE	PRI	NAME	PAGES	DIOCNT	FAULTS	CPU TIME
00000041	HIB	16	SWAPPER	0/0	0	0	00:00:01.7
00000044	HIB0	13	LANACP	1/29		SWAPPED OUT	
00000046	HIB0	10	IPCACP	0/20		SWAPPED OUT	
00000047	HIB0	9	ERRFMT	0/30		SWAPPED OUT	
00000048	HIB0	7	OPCOM	0/27		SWAPPED OUT	
00000049	HIB	10	AUDIT_SERVER	1/43	33	265	00:00:00.2
0000004A	HIB0	9	JOB_CONTROL	0/24		SWAPPED OUT	
0000004B	HIB0	9	QUEUE_MANAGER	0/33		SWAPPED OUT	
0000004C	HIB	10	SECURITY_SERVER	8/97	25	1064	00:00:01.5
0000004D	LEF0	5	DNS\$ADVER	0/37		SWAPPED OUT	
0000004E	HIB0	8	LESSACP_V30	0/32		SWAPPED OUT	
0000004F	HIB0	8	REMACP	0/16		SWAPPED OUT	
00000050	PFW	4	NET\$ACP	2/29	34	738	00:00:00.3
00000051	HIB0	4	NET\$EVD	0/29		SWAPPED OUT	
00000052	LEF0	11	DTSS\$CLERK	0/30		SWAPPED OUT	
00000053	HIB0	8	TP_SERVER	0/29		SWAPPED OUT	
00000055	HIB	8	DECW\$SERVER_0	28/328	451	1656	00:00:07.7
00000056	LEF	4	DTSESSION	8/35	301	9592	00:00:04.1
0000005A	LEF	4	DTWM	12/35	468	135296	00:00:20.1
0000005C	LEF	4	DECW\$TE_005C	8/35	116	62643	00:00:10.1
0000005D	CUR	5	_FTA1:	5/35	39	2178	00:00:01.0
0000005E	LEF	4	SYSTEM	23/64	48	896	00:00:00.7

OpenVMS Monitor Utility
PAGE MANAGEMENT STATISTICS
on node ELLIS
7-SEP-1999 00:10:08.73

	CUR	AVE	MIN	MAX
Page Fault Rate	565.01	726.51	319.23	1706.55
Page Read Rate	80.52	112.55	80.52	150.16
Page Read I/O Rate	31.02	44.75	31.02	59.40
Page Write Rate	2.97	16.93	0.00	96.47
Page Write I/O Rate	0.33	0.43	0.00	1.92
Free List Fault Rate	109.24	193.12	47.96	516.06
Modified List Fault Rate	326.73	407.68	166.98	970.49
Demand Zero Fault Rate	6.27	6.43	0.00	17.86
Global Valid Fault Rate	91.74	74.02	21.15	180.65
Wrt In Progress Fault Rate	0.00	0.00	0.00	0.00
System Fault Rate	0.00	0.10	0.00	0.62
Free List Size	65.00	127.33	62.00	299.00
Modified List Size	573.00	537.00	378.00	669.00

Process Count: 22 OpenVMS Monitor Utility Uptime: 0 00:07:37
PROCESSES
on node ELLIS
7-SEP-1999 00:10:11.81

PID	STATE	PRI	NAME	PAGES	DIOCNT	FAULTS	CPU TIME
00000041	HIB	16	SWAPPER	0/0	0	0	00:00:01.8
00000044	HIBO	13	LANACP	1/29		SWAPPED OUT	
00000046	HIBO	10	IPCACP	0/20		SWAPPED OUT	
00000047	HIBO	9	ERRFMT	0/30		SWAPPED OUT	
00000048	HIBO	7	OPCOM	0/27		SWAPPED OUT	
00000049	HIB	10	AUDIT_SERVER	1/43	33	265	00:00:00.2
0000004A	HIBO	9	JOB_CONTROL	0/24		SWAPPED OUT	
0000004B	HIBO	9	QUEUE_MANAGER	0/33		SWAPPED OUT	
0000004C	HIB	10	SECURITY_SERVER	8/97	25	1064	00:00:01.5
0000004D	LEFO	5	DNS\$ADVER	0/37		SWAPPED OUT	
0000004E	HIBO	8	LES\$ACP_V30	0/32		SWAPPED OUT	
0000004F	HIBO	8	REMACP	0/16		SWAPPED OUT	
00000050	HIBO	4	NET\$ACP	0/27		SWAPPED OUT	
00000051	HIBO	4	NET\$EVD	0/29		SWAPPED OUT	
00000052	LEFO	11	DTSS\$CLERK	0/30		SWAPPED OUT	
00000053	HIBO	8	TP_SERVER	0/29		SWAPPED OUT	
00000055	HIB	8	DECW\$SERVER_0	30/347	451	1669	00:00:07.7
00000056	LEFO	4	DTSESSION	0/27		SWAPPED OUT	
0000005A	LEFO	4	DTWM	1/27		SWAPPED OUT	
0000005C	PFW	4	DECW\$TE_005C	24/64	116	63756	00:00:10.3
0000005D	CUR	4	_FTA1:	5/34	40	2722	00:00:01.2
0000005E	LEFO	4	SYSTEM	3/26		SWAPPED OUT	

OpenVMS Monitor Utility
 PAGE MANAGEMENT STATISTICS
 on node ELLIS
 7-SEP-1999 00:10:15.16

	CUR	AVE	MIN	MAX
Page Fault Rate	4842.21	1243.75	319.23	4842.21
Page Read Rate	115.86	116.82	80.52	150.16
Page Read I/O Rate	43.71	45.64	31.02	59.40
Page Write Rate	2.99	19.83	0.00	96.47
Page Write I/O Rate	0.29	0.56	0.00	1.92
Free List Fault Rate	1748.80	392.49	47.96	1748.80
Modified List Fault Rate	2883.23	716.78	166.98	2883.23
Demand Zero Fault Rate	9.28	7.74	0.00	17.86
Global Valid Fault Rate	154.79	80.40	21.15	180.65
Wrt In Progress Fault Rate	0.00	0.00	0.00	0.00
System Fault Rate	0.00	0.08	0.00	0.62
Free List Size	109.00	118.87	62.00	299.00
Modified List Size	508.00	528.37	378.00	669.00

Process Count: 22 OpenVMS Monitor Utility Uptime: 0 00:07:44

PROCESSES

on node ELLIS

7-SEP-1999 00:10:18.19

PID	STATE	PRI	NAME	PAGES	DIOCNT	FAULTS	CPU TIME
00000041	HIB	16	SWAPPER	0/0	0	0	00:00:01.8
00000044	HIBO	13	LANACP	1/29		SWAPPED OUT	
00000046	HIBO	10	IPCACP	0/20		SWAPPED OUT	
00000047	HIBO	9	ERRFMT	0/30		SWAPPED OUT	
00000048	HIBO	7	OPCOM	0/27		SWAPPED OUT	
00000049	HIB	10	AUDIT_SERVER	1/43	33	265	00:00:00.2
0000004A	PFW	8	JOB_CONTROL	0/35	28	168	00:00:00.1
0000004B	HIBO	9	QUEUE_MANAGER	0/33		SWAPPED OUT	
0000004C	HIB	10	SECURITY_SERVER	7/83	25	1074	00:00:01.5
0000004D	LEFO	5	DNS\$ADVER	0/37		SWAPPED OUT	
0000004E	HIBO	8	LESS\$ACP_V30	0/32		SWAPPED OUT	
0000004F	HIBO	8	REMACP	0/16		SWAPPED OUT	
00000050	HIBO	4	NET\$ACP	0/27		SWAPPED OUT	
00000051	HIBO	4	NET\$EVD	0/29		SWAPPED OUT	
00000052	LEFO	11	DTSS\$CLERK	0/30		SWAPPED OUT	
00000053	HIB	8	TP_SERVER	6/53	12	521	00:00:00.4
00000055	HIB	6	DECW\$SERVER_0	27/351	451	1677	00:00:07.8
00000056	PFW	4	DTSESSION	6/35	301	9602	00:00:04.1
0000005A	PFW	4	DTWM	18/64	472	135740	00:00:20.3
0000005C	LEF	4	DECW\$TE_005C	26/64	116	79399	00:00:11.6
0000005D	CUR	4	_FTA1:	26/64	40	3179	00:00:01.3
0000005F	LEF	4	_FTA2:	33/64	19	168	00:00:00.2

\$

Based on an earlier discussion, the following represents the calculation for reasonable working set list sizes.

\$ ana/sys

OpenVMS (TM) Alpha system analyzer

SDA> ev @pms\$gl_proccntmax

Hex = 00000000.00000016 Decimal = 22

DYN\$_CWPS_TERMIN

DYN\$_FDT_CONTEXT

DYN\$_IPC_POOL

DYN\$_LM_LMTE

DYN\$_MVL

|

(remaining symbols suppressed by default)

SDA> ev @pfns\$gl_phypgcnt

Hex = 00000000.00000553 Decimal = 1363

BUG\$_DISKCLASS+00003

SDA>

SDA> ev @pfns\$gl_phypgcnt/@pms\$gl_proccntmax

Hex = 00000000.0000003D Decimal = 61

DYN\$_XWB

SDA> exit

\$

The following modifications to DECW\$GETPARAMS.COM were required to implement the changes.

\$ diff decw\$getparams.com;2 decw\$getparams.com;1

File SYS\$COMMON:[SYSMGR]DECW\$GETPARAMS.COM;2

109 \$ parameter = "PQL_MWSQUOTA/1024////////"

110 \$ GoSub CHECK_MIN

111 \$ parameter = "PQL_MWSEXTENT/4096////////"

112 \$ GoSub CHECK_MIN

File SYS\$COMMON:[SYSMGR]DECW\$GETPARAMS.COM;1

109 \$ parameter = "PQL_MWSQUOTA/2048////////"

110 \$ GoSub CHECK_MIN

111 \$ parameter = "PQL_MWSEXTENT/8192////////"

112 \$ GoSub CHECK_MIN

Number of difference sections found: 1

Number of difference records found: 3

DIFFERENCES /IGNORE=()/MERGED=1-

 SYS\$COMMON:[SYSMGR]DECW\$GETPARAMS.COM;2-

 SYS\$COMMON:[SYSMGR]DECW\$GETPARAMS.COM;1

\$

Workstation Parameter Restrictions

At boot time, workstations are required to have certain minimum parameter settings through the file: **SYSS\$MANAGER:DECW\$CHECK_PARAMS.COM** (currently in **decw\$getparams.com**).



Note

Compaq recommends that you do **not** change **SYSS\$MANAGER:DECW\$CHECK_PARAMS.COM**.

Workstation requirements have caused potential response time problems for a general multi-user computing environment.

Consider the following:

- **BORROWLIM** is set to give out loans while the **SWAPPER** is reclaiming memory.
- Loans are granted up to 1/4 to 1/2 of available memory.
- Page size is 8192, so 32MB translates to 4096.
- **WSDEC** and **WSINC** are set to large values, causing potentially severe memory reclamation in a CPU bound and memory tight environment.

Locality and Paging Performance

Module 6

Module Outline

Objectives

This module describes the effects of locality on paging performance. Locality is the grouping of code or data in close proximity of access either through time or in memory. Students will be able to describe the factors associated with locality and how they affect paging performance and memory requirements on a system.

Topics

- Locality
- Locality Issues
- Effects of Poor Locality
- Effect of Improving Locality

Locality

The access of common data or code in close proximity in virtual memory is referred to as locality.

- *Spatial locality* occurs when data or code is close together in virtual memory.
- An array is an example of entity with good spatial locality as array elements are adjacent in memory.
- *Temporal locality* occurs when the same memory items are accessed at roughly the same time.
- A modular program, generally, has good locality as the program and its subroutines are accessed closely in time, although they may be spread through virtual memory.
- An array, accessed randomly, may have poor temporal locality.
- From the viewpoint of paging performance, temporal locality is more important than spatial locality.
- Because code usually flows sequentially, it has inherent temporal locality.
- Data, however, is easy to access with poor locality. A programmer should take great care in designing data access with the best locality possible.

Locality Issues

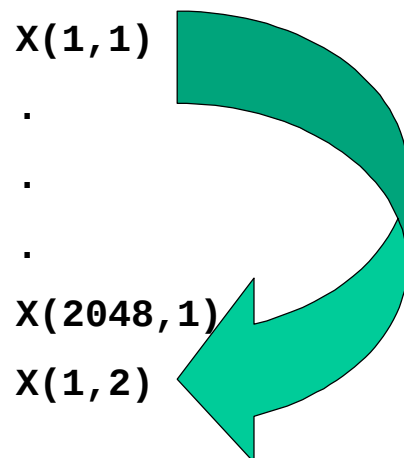
One major issue in locality is access of data in multi-dimensional arrays.

**Note**

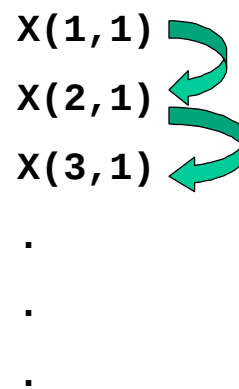
FORTRAN is a column order language (data is stored with leftmost index changing most frequently).

Most other languages (COBOL, PASCAL, BASIC, C, PL/I, and so on) are row order (rightmost index ordered).

Accessing a column order array
in row order:



Accessing a column order array
in column order:



- Poor locality can be caused by traversing an array "unnaturallly", touching only one element in the page before moving to the next page.
 - "Natural" access causes touching all data items before moving to the next page.
- Another example of poor locality is the mechanism used extensively in traditional **FORTRAN** programs.
 - When a **FORTRAN** programmer needed multiple items to describe a data element and there were several data items, the programmer typically set up an array for each item.
 - Then, the programmer used the same index into each array to obtain items for given element.
 - Assuming arrays were large enough, each item could have been stored in different pages, causing poor locality in access of the data.
 - A better approach is to use structure constructs. Most high level languages, even **FORTRAN**, support some form of structure construct.

Importance of Locality

- An application with poor locality will be touching more, possibly many more pages than an application written with good locality.
 - By touching more pages, the program will most likely generate many more page faults.
 - Poorly designed applications straddle the *performance manager* with the task of increasing working sets and possibly purchasing more memory.
 - Spending more time and care in good application design yields less effort and expense in supporting system performance characteristics.

Effects of Poor Locality

In this first example, we are running a **FORTRAN** program written with poor locality; working set sizes are all set to **300** pagelets.

- Elapsed CPU time for the run is $2:10.95 - 0:01.02 = 2:09.93$.
- Total number of faults for the run is $529296 - 481 = 528415$.

```
$
$ set work/limit=300/quota=300/extent=300
$ show status
  Status on 25-OCT-1999 11:15:53.81      Elapsed CPU : 0 00:00:01.02
  Buff. I/O :      54      Cur. ws. : 300      Open files :      0
  Dir. I/O :      35      Phys. Mem. : 300      Page Faults :      481
$ run thrasher
$ show status
  Status on 25-OCT-1999 11:25:10.10      Elapsed CPU : 0 00:02:10.95
  Buff. I/O :      60      Cur. ws. : 300      Open files :      0
  Dir. I/O :      36      Phys. Mem. : 153      Page Faults : 529296
$
```

The second run with 1000 working set list sizes shows little improvement.

- Total CPU time is 3:59.61 - 2:11.12 = 1:48.49.
- Total faults are 1053795 - 529375 = 524420.
- Memory in this case was wasted.

```
$
$ set work/limit=1000/quota=1000/extent=1000
$ show status
Status on 25-OCT-1999 11:27:39.61      Elapsed CPU : 0 00:02:11.12
Buff. I/O :      79      Cur. ws. : 1000      Open files :      0
Dir. I/O :      36      Phys. Mem. : 196      Page Faults : 529375
$ run thrasher
$ show status
Status on 25-OCT-1999 11:29:36.80      Elapsed CPU : 0 00:03:59.61
Buff. I/O :      93      Cur. ws. : 1000      Open files :      0
Dir. I/O :      37      Phys. Mem. : 155      Page Faults : 1053795
$
```

In the third run with working set list sizes of 1500 pages, numbers did improve substantially.

- Total CPU time is $4:02.89 - 3:59.75 = 3.14$.
- Total faults are $1058088 - 1053874 = 4214$.

```
$
$ set work/limit=1500/quota=1500/extent=1500
$ show status
Status on 25-OCT-1999 11:34:39.77      Elapsed CPU : 0 00:03:59.75
Buff. I/O :      108      Cur. ws. : 1500      Open files :      0
Dir. I/O :      37      Phys. Mem. : 200      Page Faults : 1053874
$ run thrasher
$ show status
Status on 25-OCT-1999 11:34:51.16      Elapsed CPU : 0 00:04:02.89
Buff. I/O :      114      Cur. ws. : 1500      Open files :      0
Dir. I/O :      38      Phys. Mem. : 156      Page Faults : 1058088
$
$
```

Effect of Improving Locality

The **FORTRAN** program **THRASHER.FOR** (written in column order language) references array in row order.

- When rewritten using proper indexing (**NOT_MY_FAULT.FOR**), the program runs substantially faster, with much less memory allotted.
- Because it runs with better or equal performance with a working set list size of **300** as compared to **1500** pages, we could have five times as many users running the program in the same memory configuration.
- Total CPU time for run is $4:05.78 - 4:03.05 = 2.73$.
- Total faults are $1062423 - 1058167 = 4236$.

Note

Most other languages are row order languages, and the improper mechanism in **FORTRAN** yields better locality and performance in those languages. Check the language-specific user's manual to determine the mechanism in array storage.

```
$
$ set work/limit=300/quota=300/extent=300
$ show status
Status on 25-OCT-1999 11:35:18.47      Elapsed CPU : 0 00:04:03.05
Buff. I/O :      131      Cur. ws. : 300      Open files :      0
Dir. I/O :      38      Phys. Mem. : 200      Page Faults : 1058167
$ run not_my_fault
$ show status
Status on 25-OCT-1999 11:35:30.96      Elapsed CPU : 0 00:04:05.78
Buff. I/O :      137      Cur. ws. : 300      Open files :      0
Dir. I/O :      39      Phys. Mem. : 154      Page Faults : 1062423
$
```

Sample FORTRAN Programs

```
$ type thrasher.for
integer      array(512,1024)
common array
do i=1,512
  do j=1,1024
    array(i,j) = 42
  enddo
enddo
end

$ type not_my_fault.for
integer      array(512,1024)
common array
do j=1,1024
  do i=1,512
    array(i,j) = 42
  enddo
enddo
end

$
```

C Locality Programs

```
$ type thrasher.c
#include <stdlib.h>
int main(void)
{
    int arr[1000][1000];
    int i,j;

    for(j=0;j<1000;j++)
        for(i=0;i<1000;i++)
            arr[i][j] = 42;
    return(EXIT_SUCCESS);
}
$ sh stat
Status on 7-SEP-1997 02:02:57.40 Elapsed CPU : 0 00:00:21.71
Buff. I/O : 4125 Cur. ws. : 1024 Open files : 0
Dir. I/O : 3224 Phys. Mem. : 928 Page Faults : 162105
$ r thrasher
$ sh stat
Status on 7-SEP-1997 02:03:25.28 Elapsed CPU : 0 00:00:36.82
Buff. I/O : 4137 Cur. ws. : 1024 Open files : 0
Dir. I/O : 3227 Phys. Mem. : 976 Page Faults : 661263
$
```

Note

These C examples were compiled under an older compiler. The current compiler would optimize away all of the code. See the lab exercises for methods of bypassing this optimization.

```
$ typ not_my_fault.c
#include <stdlib.h>
int main(void)
{
    int arr[1000][1000];
    int i,j;

    for(i=0;i<1000;i++)
        for(j=0;j<1000;j++)
            arr[i][j] = 42;
    return(EXIT_SUCCESS);
}
$ sh stat
Status on 7-SEP-1997 02:05:34.93 Elapsed CPU : 0 00:00:44.82
Buff. I/O : 4415 Cur. ws. : 1024 Open files : 0
Dir. I/O : 3564 Phys. Mem. : 976 Page Faults : 751983
$ r not_my_fault
$ sh stat
Status on 7-SEP-1997 02:05:43.28 Elapsed CPU : 0 00:00:45.43
Buff. I/O : 4427 Cur. ws. : 1024 Open files : 0
Dir. I/O : 3566 Phys. Mem. : 1024 Page Faults : 753054
$
```

Shared Image Costs and Benefits

Module 7

Module Outline

Objectives

This module describes the costs and benefits associated with images that have been installed with the **/SHARE** qualifier. Upon completion of this module, students should be able to describe the benefits of installing images **/SHARE**, both from the standpoint of conserving memory and speeding up the process of image activation. Resident Images on Alpha systems will also be discussed.

Topics

- Shared Image Costs and Benefits
- Image Activation and Paging
- Linking Alpha Shareable Images
- Alpha Resident Images
- Shared Address Data
- Shared and Other Install Options

Shared Image Costs and Benefits

Shared Image Costs

Installing image **/SHARE** does not make image memory resident.

- Installing an image does not tie up memory unnecessarily for each page of the image.
- Cost for installing image **/SHARE** is made up of the size of data structures required to support global paging.
- Structures are global section descriptors, global page table entries, and global section table entries.

Global Section Descriptors

Global section descriptors describe name, characteristics, and maintain pointers to other structures.

- Allocated out of paged pool and take up a few hundred bytes per structure.
- Remember: paged pool is pageable, so cost is fairly insignificant.

Global Section Table Entries

- Allow the pager to locate sections of code from the shared image file on disk.
- 32 bytes in size (VAX)/40 bytes (Alpha) and are nonpageable.
- To calculate costs, multiply the number of sections required by image by size.
- Cost is in bytes of nonpageable memory permanently allocated to OpenVMS.
- **GBLSECTIONS** global section table entries are preallocated on the system.

Global Page Table Entries

- Identify the current location of global page in memory or on disk.
- Pageable – the only cost is nonpageable bytes of the system page table entries required to map pages in the section.
- 4 bytes in size (VAX)/8 bytes (Alpha) - 128 in a page (VAX)/1024 in a page (Alpha).
- OpenVMS requires one system page table entry to map a page of system space.
- System page table entries are 4 bytes in size – the cost for global page table entries is the number of global page table entries in the section times 4 divided by 128, or more simply divided by 32 (VAX). Alpha 8/1024 == 1/128.
- The number of global page table entries preallocated on the system is defined by **SYSGEN** parameter **GBLPAGES**.

Global Paging Benefits

- An obvious benefit to global paging by installing image **/SHARE** is that there may be some sharing of memory, which minimizes memory requirements and improves quality of paging (**GLOBAL VALID FAULT** rate, **FREE LIST FAULT** rate, and for writable global sections, **MODIFIED LIST FAULT** rate).
- A less obvious benefit to global paging is that when an image runs down, global pages go to the end of the free page list and the global page table entries still describe them.
- The next activation of image has a chance (with large free list , a good chance) to locate pages from the image in memory.
- Yields faster image activation, even if pages are not actually being shared simultaneously.
- Implies frequently shared images should be installed **/SHARE** due to the small cost of having them installed and additional benefit of faster image activation.

Image Activation and Paging

Page Fault Rate for Images Installed /SHARE

Do not be surprised to experience a higher page fault rate for images installed /**SHARE** as there is no prefetching on these soft faults.

The following is a sample from **\$MONITOR PAGE** obtained while running a program that was made up of a large read-only code segment over and over.



Note

The hard fault rate is averaging 9.96 per second and page read rate is averaging 447.31.

- Subtracting the hard fault rate from the total fault rate (64.87), you will find that 54.91 pages are being made valid per second due to soft faulting.
- Page read rate defines the number of pages being made valid due to hard faults.
- Summing the number of pages being made valid due to hard and soft faults = 502.22 pages being made valid per second on these sample runs.

VAX/VMS Monitor Utility

PAGE MANAGEMENT STATISTICS				
on node BRUDEN				
2-AUG-1999 12:08:48				
	CUR	AVE	MIN	MAX
Page Fault Rate	78.59	64.87	51.41	78.59
Page Read Rate	448.88	447.31	445.76	448.88
Page Read I/O Rate	9.90	9.96	9.90	10.03
Page Write Rate	0.00	0.00	0.00	0.00
Page Write I/O Rate	0.00	0.00	0.00	0.00
Free List Fault Rate	1.91	1.42	0.94	1.91
Modified List Fault Rate	38.01	31.17	24.45	38.01
Demand Zero Fault Rate	2.23	2.37	2.23	2.50
Global Valid Fault Rate	26.19	19.46	12.85	26.19
Wrt In Progress Fault Rate	0.00	0.00	0.00	0.00
System Fault Rate	0.00	0.00	0.00	0.00
Free List Size	2795.00	2680.00	2565.00	2795.00
Modified List Size	142.00	156.00	142.00	170.00

Shared Image Benefits

This sample **\$MONITOR PAGE** display comes from running the same program from the previous page over and over.

- Difference = program has now been installed **/SHARE**.
- Note that the hard fault rate, on average, dropped from 9.96 to 6.68 per second.
- Soft fault rate (total-hard) has gone up significantly from the previous page to 1696.24 per second from 54.91.
- Soft faults are being resolved primarily from the free page list (1514.50 per second).
- After installing image **/SHARE**, global pages containing code are placed at the end of the free list at image rundown.
- Allows for soft faulting, which is occurring during image activation process in this display.

VAX/VMS Monitor Utility
PAGE MANAGEMENT STATISTICS
on node BRUDEN
2-AUG-1999 12:09:44

	CUR	AVE	MIN	MAX
Page Fault Rate	1706.85	1702.92	1680.69	1720.87
Page Read Rate	6.54	8.14	6.54	11.21
Page Read I/O Rate	6.54	6.68	6.54	6.85
Page Write Rate	0.00	0.00	0.00	0.00
Page Write I/O Rate	0.00	0.00	0.00	0.00
Free List Fault Rate	1527.41	1514.50	1480.69	1534.89
Modified List Fault Rate	98.13	102.60	98.13	107.59
Demand Zero Fault Rate	6.54	6.78	6.54	7.16
Global Valid Fault Rate	66.35	70.66	66.35	77.53
Wrt In Progress Fault Rate	0.00	0.00	0.00	0.00
System Fault Rate	0.00	0.00	0.00	0.00
Free List Size	2910.00	2913.00	2910.00	2919.00
Modified List Size	137.00	145.66	137.00	158.00

Shareable Image Characteristics

A shareable image is a linked image file that retains its global symbols and may be linked against. Shareable images are mapped at run time automatically by the Image Activator.

- Shareable images can be installed (**\$INSTALL**) with the **/SHARE** qualifier to enable potential sharing of code/data (writable psects with the **SHR** attribute will be shared).
- OpenVMS allows for the creation of shareable images, which are upwardly compatible.
- Characteristics of OpenVMS shareable images:
 - Saves disk space; image code is shared on disk.
 - Saves on memory requirements; code in memory is shared.
 - Speeds up the image activation process, because images are likely to be resident if the free page list is sufficiently large.
 - Can change the code in the shareable image without having to relink applications that originally linked to the shareable image.

Linking Alpha Shareable Images

- When creating a shareable image, use the **/SHARE=*image*** qualifier on the command line.
- Include an options file, using **/OPTIONS=*file.opt***.
- The options file should minimally contain the **GSMATCH** option to identify version and upgradeability characteristics of the shareable image. **GSMATCH** format:

`GSMATCH = major_id,minor_id, [EQUAL, LEQUAL, ALWAYS]`

- To link against a shareable image, you must specify the shareable image in the linker options file. Example:

```
$ LINK MAIN,SY$INPUT/OPTIONS
```

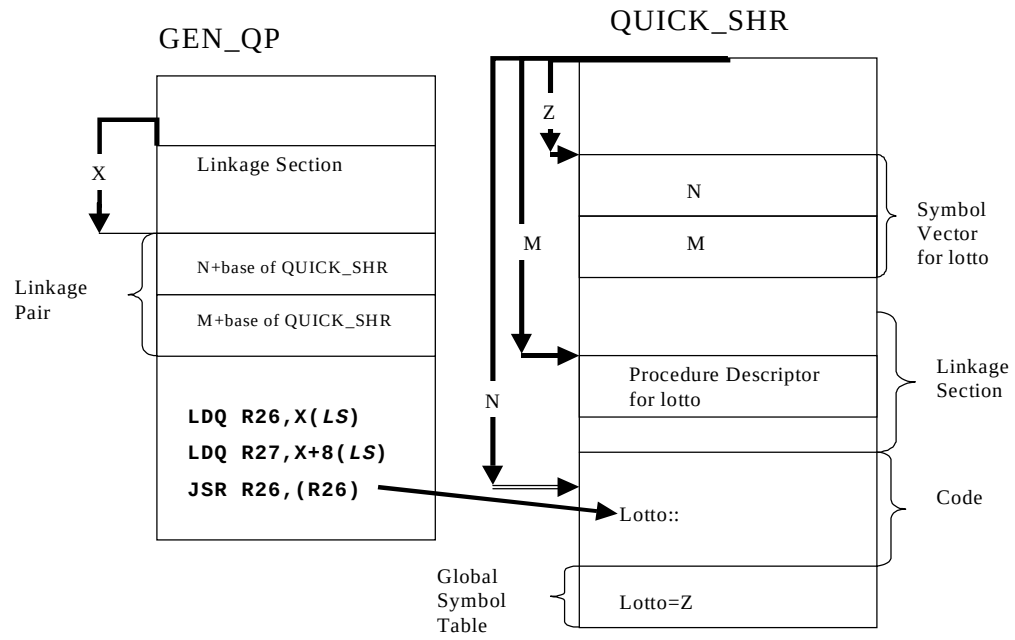
```
shareable_image/SHARE
```

```
^Z
```

```
$
```

- You can produce a shareable image library using the Librarian Utility (**\$LIBRARY**) with the **/SHARE** qualifier. A shareable image library acts in similar fashion to an object library, in that it contains global symbols and can be linked against. However, the code/data for the shareable image is not moved into the library, only a *pointer* to the shareable image file.
- The Image Activator looks in the directory **SY\$SHARE:** for the shareable image, by default. You can create a logical name (corresponding to the shareable image file name) to get the Image Activator to look elsewhere.

Alpha Shareable Images



Alpha Shareable Images Example

```
$
$ type quick_pik.c
void lotto(int *);
int get_ball(int, int *);
int is_member(int,int *);
int *quick_pick(int *,int);
int *sort_set(int *);
void print(int *);

#include <stdio.h>
#include <stdlib.h>
#include <time.h>
static const int BALLS_PER_DRAW = 6;
static const int FIRST_BALL = 0;
static const int LOW_BALL = 1;

/* Lotto initializer. */
void lotto(int *ball)
{
    int i;
    for(i=0;i<BALLS_PER_DRAW;i++)
        ball[i] = 0;
}
/* Function to return a ball value, returns 0 if invalid ball */
int get_ball(int val, int *ball)
{
    if((val>=0) && (val < BALLS_PER_DRAW))
    {
        return(ball[val]);
    }
    return(0);
}
/* Function to determine whether a value is currently a
   member of the lotto set. Returns Ball value if it is a member, returns 0
   if not found.
*/
```

```
int  is_member(int val,int *ball)
{
    int    i;
    for(i=0;i<BALLS_PER_DRAW;i++)
    {
        if(val == ball[i])
        {
            return(val);
        }
    }
    return(0);
}

/* Generate a lotto quick pick using rand */
const int shift_val = 7;
int  *quick_pick(int *ball,int max_ball)
{
    int    i;
    int    temp;
    srand((int)clock()); /* Set the seed based on the current time*/
    for(i = 0; i < BALLS_PER_DRAW; i++)
    {
        do
        {
            /* Generate a relatively random lotto number */
            temp = ((rand())>>shift_val)%max_ball + 1;
        } while(is_member(temp,ball)); /* If already drawn try again
        /* Set ball */
        ball[i] = temp;
    }
    /* return the quick pick set */
    return(ball);
}
```

```
/* Sort the lotto numbers */
int *sort_set(int *ball)
{
    int    temp_ball;
    int    switches;
    int    i,j;

    for(i = 0; i < BALLS_PER_DRAW; i++)
    {
        switches = FALSE;
        for(j = 0; j < BALLS_PER_DRAW-1; j++)
        {
            /* Swap if previous is greater */
            if(ball[j]>ball[j+1])
            {
                temp_ball = ball[j];
                ball[j] = ball[j+1];
                ball[j+1] = temp_ball;
                switches = TRUE;
            }
        }
        /* If no swaps the list is sorted */
        if(!switches) break;
    }
    /* Return sorted set*/
    return(ball);
}

/* Function to print the current lotto set */
void qp_print(int *ball)
{
    int    i;
    sort_set(ball);
    printf("Your LUCKY numbers are: ");
    for(i = FIRST_BALL;i<BALLS_PER_DRAW;i++)
        printf("%d ",ball[i]);
    putchar('\n');
}
$
```

```

$ !BUILD AN AXP UPWARDLY COMPATIBLE SHAREABLE IMAGE
$ type build_qp_shr.com
$ cc quick_pik
$ link/shareable=quick.exe quick_pik,sys$input/OPTIONS
GSMATCH=LEQUAL,1,1
SYMBOL_VECTOR=(lotto=PROCEDURE,-
               quick_pick=PROCEDURE,-
               sort_set=PROCEDURE,-
               qp_print=PROCEDURE)

$
$ set ver
$
$ @build_qp_shr
$ cc quick_pik
$ link/shareable=quick.exe quick_pik,sys$input/OPTIONS
GSMATCH=LEQUAL,1,1
SYMBOL_VECTOR=(lotto=PROCEDURE,-
               quick_pick=PROCEDURE,-
               sort_set=PROCEDURE,-
               qp_print=PROCEDURE)

$
$ set proc/priv=all
$ define/system quick dka300:[ellis.shr]quick
%DCL-I-SUPERSEDE, previous value of QUICK has been superseded
$
$ install add quick/share/header
$
$ install list quick/global/full

DISK$AXPVMSSYS:<ELLIS.SHR>.EXE
    QUICK;2          Open Hdr Shar          Lnkbl
    Entry access count      = 0
    Current / Maximum shared = 1 / 0
    Global section count    = 2

    System Global Sections

QUICK_002      (01000001)          PRM SYS      Pgltcnt/Refcnt=3/0
QUICK_001      (01000001)          PRM SYS      Pgltcnt/Refcnt=1/0
              Owner:      [ELLIS]
              Protection:  S:RWED,O:RWED,G:RE,W:

$

```

```
$ type gen_qp.c
#include <stdio.h>
#include <stdlib.h>
#define BALLS_PER_DRAW 6

int main(void)
{
    int    lotto_set[BALLS_PER_DRAW];
    int    max_ball = 42;
    void    lotto(int *);
    int    *quick_pick(int *,int);
    void    qp_print(int *);

    lotto(lotto_set);
    quick_pick(lotto_set,max_ball);
    qp_print(lotto_set);

    return(EXIT_SUCCESS);
}
$ cc gen_qp
$
$ link gen_qp,sys$input/option
quick/share
^Z
$
$ run gen_qp
Your LUCKY numbers are: 4 11 13 29 31 35
$
$ edit/edt quick_pik.c
      1      void lotto(int *);
*'LUCKY'
      99      printf("Your LUCKY numbers are: ");
*s/LUCKY/winning/
      99      printf("Your winning numbers are: ");
1 substitution
*exit
_DKA300:[ELLIS.SHR]QUICK_PIK.C;12 103 lines
$
```

```
$ @build_qp_shr
$cc quick_pik
$link/shareable=quick.exe quick_pik,sys$input/OPTIONS
GSMATCH=LEQUAL,1,1
SYMBOL_VECTOR=(lotto=PROCEDURE,-
               quick_pick=PROCEDURE,-
               sort_set=PROCEDURE,-
               qp_print=PROCEDURE)

$
$
$ install replace quick/share/header
$
$ run gen_qp
Your winning numbers are: 2 9 17 19 20 40
$
```

Alpha Resident Images

Alpha systems allow shareable images that have been linked properly to be installed with the **/RESIDENT** qualifier.

- Resident images are more efficient than shareable images, because they are guaranteed to be memory resident and they are mapped into *huge* pages, allowing for more effective utilization of the AXP translation buffer (TB). (One page implies one TB entry is required to map the page, increasing TB hit rates.)
- For an image to be installed **/RESIDENT** it must be *sliced* and sorted into sections with the same attributes and protection. These sections are installed into huge pages called granularity hint regions (**GHRs**) in memory. The AXP hardware maps these pages as a single **GHR**.
- Resident images for user programs require that you reserve **GHR** pages for use after boot, using the **SYSGEN** parameter **GH_RSRVPGCNT**.
- Code sections of resident images are mapped into system space.

Section Binding

Shareable and executable images can be installed resident if they have been linked properly.

- To allow the linker to slice an image, you must use the **/SECTION_BINDING** qualifier. Format:
`/[NO]SECTION_BINDING[=(CODE,DATA)]`
- To determine whether an image was linked with this qualifier, use **\$ANALYZE/IMAGE** and look for **EIHD\$V_BIND_CODE** symbol set to 1.
- Limitations:
 - Data must remain process private and will not be moved into system space.
 - Code and data which have dependencies on the image section layout may cause bind failures.
 - The parameter **GH_RSRVPGCNT** must allow the use of **GHRs**.

Alpha Resident Images Example

```

$
$
$ type build_qp_res.com
$cc quick_pik
$link/notrace/shareable=quick_res.exe/section_binding quick_pik, -
sys$input/OPTIONS

GSMATCH=LEQUAL,1,1
SYMBOL_VECTOR=( lotto=PROCEDURE, -
                 quick_pick=PROCEDURE, -
                 sort_set=PROCEDURE, -
                 qp_print=PROCEDURE)

$
$
$
$ set ver
$
$ @build_qp_res
$cc quick_pik
$link/notrace/shareable=quick_res.exe/section_binding quick_pik, -
sys$input/OPTIONS

GSMATCH=LEQUAL,1,1
SYMBOL_VECTOR=( lotto=PROCEDURE, -
                 quick_pick=PROCEDURE, -
                 sort_set=PROCEDURE, -
                 qp_print=PROCEDURE)

$
$ set proc/priv=all
$
$ define/system quick_pik dka300:[ellis.shr]quick_res

```

```

$
$ show memory
      System Memory Resources on 25-AUG-1999 14:16:04.30
Physical Memory Usage (pages):   Total      Free      In Use   Modified
Main Memory (32.00Mb)           4096      689      2860     547
Virtual I/O Cache (Kbytes):     Total      Free      In Use
Cache Memory                     3200       0      3200
Granularity Hint Regions (pages): Total      Free      In Use   Released
Execlet code region              512       0      286     226
Execlet data region              128       0       64      64
VMS exec data region             144       1     143       0
Resident image code region       512     200     263      49
Slot Usage (slots):             Total      Free   Resident   Swapped
Process Entry Slots              48      31        14       3
Balance Set Slots                46      31        12       3
Dynamic Memory Usage (bytes):    Total      Free      In Use   Largest
Nonpaged Dynamic Memory          999424   459712   539712   421696
Paged Dynamic Memory             802816   423808   379008   422912
Paging File Usage (blocks):      Free   Reservable   Total
DISK$AXPVMSSYS:[SYS0.SYSEXE]SWAPFILE.SYS  2944     2944     6016
DISK$AXPVMSSYS:[SYS0.SYSEXE]PAGEFILE.SYS  66704    29648    79872
Of the physical pages in use, 1902 pages are permanently allocated to VMS.
$
$ mcr sysgen
SYSGEN> SH GH
Parameter Name      Current   Default   Min.     Max.     Unit
Dynamic
-----
-----
GH_EXEC_CODE        512      512       0      2048 Pages
GH_EXEC_DATA        128      128       0      2048 Pages
GH_RES_CODE         512      512       0      2048 Pages
GH_RES_DATA         0         0         0      2048 Pages
GH_RSRVPGCNT        200       0         0      2048 Pages
SYSGEN>
SYSGEN> EXIT
$
$ install add quick_res/resident
$

```

```
$ install list quick_res/full
```

```
DISK$AXPVMSSYS:<ELLIS.SHR>.EXE
```

```
  QUICK_RES;4      Open Hdr Shar      Lnkbl  Resid
    Entry access count      = 0
    Current / Maximum shared = 1 / 0
    Global section count    = 1
    Resident section count  = 0001
```

```
$
```

```
$
```

```
$ install list quick_res/full/glob/resident
```

```
DISK$AXPVMSSYS:<ELLIS.SHR>.EXE
```

```
  QUICK_RES;4      Open Hdr Shar      Lnkbl  Resid
    Entry access count      = 0
    Current / Maximum shared = 1 / 0
    Global section count    = 1
    Resident section count  = 0001
```

Base VA	End VA	Length	Type
8060E000	8060E600	00000600	Resident Code

System Global Sections

```
QUICK_RES_001 (01000001)      PRM SYS      Pgltcnt/Refcnt=1/0
    Owner:      [ELLIS]
    Protection: S:RWED,0:RWED,G:RE,W:
```

```
$
```

\$ install list/resident

System Resident Sections

DISK\$AXPVMSSYS:<ELLIS.SHR>.EXE
QUICK_RES;4

Base VA	End VA	Length	Type
8060E000	8060E600	00000600	Resident Code

DISK\$AXPVMSSYS:<SYS0.SYSCOMMON.SYSLIB>.EXE

...

DECC\$SHR;1

Base VA	End VA	Length	Type
80554000	8060D200	000B9200	Resident Code
7FE58000	7FE71600	00019600	Linkage
7FE78000	7FE7EA00	00006A00	Writeable data
7FE88000	7FE8A200	00002200	Read-only data
7FE98000	7FE98200	00000200	Writeable data
7FEA8000	7FEAC400	00004400	Demand-zero

...

LIBOTS;1

Base VA	End VA	Length	Type
8048C000	80499C00	0000DC00	Resident Code
7FBB8000	7FBBA600	00002600	Read-only data
7FBC8000	7FBC9C00	00001C00	Linkage

...

\$

\$ show mem

System Memory Resources on 25-AUG-1999 14:18:59.38

Physical Memory Usage (pages):	Total	Free	In Use	Modified
Main Memory (32.00Mb)	4096	699	2861	536
Virtual I/O Cache (Kbytes):	Total	Free	In Use	
Cache Memory	3200	0	3200	
Granularity Hint Regions (pages):	Total	Free	In Use	Released
Execlet code region	512	0	286	226
Execlet data region	128	0	64	64
VMS exec data region	144	1	143	0
Resident image code region	512	199	264	49
Slot Usage (slots):	Total	Free	Resident	Swapped
Process Entry Slots	48	31	14	3
Balance Set Slots	46	31	12	3
Dynamic Memory Usage (bytes):	Total	Free	In Use	Largest
Nonpaged Dynamic Memory	999424	458304	541120	421696
Paged Dynamic Memory	802816	422960	379856	422240
Paging File Usage (blocks):		Free	Reservable	Total
DISK\$AXPVMSSYS:[SYS0.SYSEX]SWAPFILE.SYS		2944	2944	6016
DISK\$AXPVMSSYS:[SYS0.SYSEX]PAGEFILE.SYS		66704	29920	79872

Of the physical pages in use, 1902 pages are permanently allocated to VMS.

\$

\$

\$ link/executable=gen_qp_res gen_qp,sys\$input/option
quick_res/share

^Z

\$

\$

\$ r gen_qp_res

Your winning numbers are: 4 11 13 21 26 42

\$

\$

Alpha Sliced Executable

\$ ana/image quick_res

Analyze Image 1-SEP-1999 12:51:41.26 Page 1
DKA300:[ELLIS.SHR]QUICK_RES.EXE;6
ANALYZ A05-19

This is an OpenVMS Alpha image file

IMAGE HEADER

Fixed Header Information

image format major id: 3, minor id: 0
header block count: 2
image type: shareable (EIHD\$K_LIM)
 global section major id: %X'01', minor id: %X'000001'
 match control: ISD\$K_MATLEQ
I/O channel count: default
I/O pagelet count: default
Symbol Vector Virtual Address: %X'00000110'
Symbol Vector Size: 64 bytes
Virtual Memory Block Size: 65536 (BPAGE = 16)
Fixup Section Virtual Address: %X'00030000'
linker flags:
 (0) EIHD\$V_LNKDEBUG 0
 (1) EIHD\$V_LNKNOTFR 0
 (2) EIHD\$V_NOP0BUFS 0
 (3) EIHD\$V_PICIMG 1
 (4) EIHD\$V_P0IMAGE 0
 (5) EIHD\$V_DBGDMT 1
 (6) EIHD\$V_INISHR 0
 (7) EIHD\$V_XLATED 0
 (8) EIHD\$V_BIND_CODE 1
 (9) EIHD\$V_BIND_DATA 1

Image Activation Information

...

ANA/IMAGE QUICK_RES
\$

AXP Resident Executable Example

```
$ ana/image qp_all_res
```

```
Analyze Image
1
```

```
1-SEP-1999 12:52:17.01 Page
```

```
_DKA300:[ELLIS.SHR]QP_ALL_RES.EXE;2
ANALYZ A05-19
```

```
This is an OpenVMS Alpha image file
```

```
IMAGE HEADER
```

Fixed Header Information

```
image format major id: 3, minor id: 0
header block count: 2
image type: executable (EIHD$K_EXE)
I/O channel count: default
I/O pagelet count: default
Symbol Vector Virtual Address: %X'00000000'
Symbol Vector Size: 0 bytes
Virtual Memory Block Size: 65536 (BPAGE = 16)
Fixup Section Virtual Address: %X'00030000'
linker flags:
(0) EIHD$V_LNKDEBUG 0
(1) EIHD$V_LNKNOTFR 0
(2) EIHD$V_NOP0BUFS 0
(3) EIHD$V_PICIMG 1
(4) EIHD$V_P0IMAGE 0
(5) EIHD$V_DBGDMT 1
(6) EIHD$V_INISHR 0
(7) EIHD$V_XLATED 0
(8) EIHD$V_BIND_CODE 1
(9) EIHD$V_BIND_DATA 1
```

Image Activation Information

```
first transfer address: %X'00010050'
second transfer address: %X'00000000'
third transfer address: %X'00000000'
```

```
...
```

```
ANA/IMAGE QP_ALL_RES
$
```

Shared Address Data

OpenVMS images contain a special program section called the *linkage section* that contains constants and addresses. As well as the linkage section, a program may have read-only sections that contain addresses that require run-time fix-up operations.

- Because some of these addresses reference shareable image routines, they need to be initially writeable. The Image Activator changes the protection on these pages at image activation time to read only after "fix-up" is done on these addresses.
- Since these pages were originally modified, their data was not shared between processes.
- Fixing up addresses must be done at image activation time, even if images are not changed over time.
- Each shareable image is mapped at the lowest P0 space address available at the time it is mapped. This address depends on the main image size and other shareable images mapped.

Shared Address Data Implementation

To fix shareable image addresses, the images are assigned unique, fixed addresses in P1 space when an image is installed with the **/SHARE=ADDRESS_DATA** qualifier.

Fixing the shareable image addresses for all processes allows the sharing of the linkage section information by all processes running programs; the linkage section is created as a global section and the necessary fix-ups are made, when the image is installed.

Installing images with the **/SHARE=ADDRESS_DATA** qualifier can improve performance by:

- Minimizing CPU overhead to perform fix-up operations.
- Eliminating an I/O to access the linkage section.
- Minimizing memory requirements, through the sharing of the linkage section and other read-only sections containing addresses.

Works with:

- Shared images
- Shareable images
- Images installed resident

Limits to Shared Address Data

- If any called images are not installed with shared address data, then the calling images are not installed with shared address data.
- If any called images are deinstalled, then at image activation time the data is not shared.
- P1 space usage for shared address data is limited by the **SYSGEN** parameter **IMGREG_PAGES**. It is needed only for shareable images, not for image.
- Most OpenVMS shareable images are installed with shared address data.
- Commands to set up shared address data:

```
INSTALL> CREATE image /SHARED=[NO]ADDRESS_DATA]
INSTALL> ADD image /SHARED=[NO]ADDRESS_DATA]
INSTALL> REPLACE image /SHARED=[NO]ADDRESS_DATA]
```

- Sample images installed with shared data:

```
$ inst list sys$share:decc$shr
```

```
DISK$AXP71FT2:<SYS0.SYSCOMMON.SYSLIB>.EXE
  DECC$SHR;1      Open Hdr SharAddr      Lnkbl
Resid
```

```
$ inst list sys$share:libots
```

```
DISK$AXP71FT2:<SYS0.SYSCOMMON.SYSLIB>.EXE
  LIBOTS;1      Open Hdr SharAddr      Lnkbl
Resid
```

```
$ inst list sys$share:librtl
```

```
DISK$AXP71FT2:<SYS0.SYSCOMMON.SYSLIB>.EXE
  LIBRTL;1      Open Hdr SharAddr      Lnkbl      Resid
$ mcr sysgen SH IMGREG_PAGES
Parameter Name      Current      Default      Min.      Max.      Unit
Dynamic
-----
--
IMGREG_PAGES        10000      10000        0      50000 Pages
SYSGEN> EXIT
$
```

Shared Address Data Benefits Example

```
$ type fault.c
/* Program to display total and hard faults, since process creation.
   Program uses undocumented interfaces which neither supported
   nor recommended.
   Author: Bruce Ellis
 */
#include <stdio.h>
#include <ssdef.h>
#include <phddef.h>

extern      PHD *ctl$gl_phd;

int main(void)
{
/* Display total and hard faults. */
    printf("Total faults: %ld\n",ctl$gl_phd->phd$l_pageflts);
    printf("Hard faults:  %ld\n",ctl$gl_phd->phd$l_pgfltio);
    return(SS$NORMAL);
}
$ cc fault+sys$library:sys$lib_c/lib
$ link fault/sysexe
```

Examining image sections:

```
$ ana/image fault
...
```

Linkage section representation in an image file:

Image Section Descriptors (ISD)

```
1) image section descriptor (36 bytes)
   byte count: 512
   base virtual address: %X'00010000' (P0 space)
   page fault cluster size: default
   ISD flags:
       (0) EISD$V_GBL      0
       (1) EISD$V_CRF      1
       (2) EISD$V_DZRO     0
       (3) EISD$V_WRT      1
       (4) EISD$V_INITALCOD 0
       (5) EISD$V_BASED     0
       (6) EISD$V_FIXUPVEC  0
       (7) EISD$V_RESIDENT  0
       (8) EISD$V_VECTOR    0
       (9) EISD$V_PROTECT   0
      (10) EISD$V_LASTCLU   1
      (11) EISD$V_EXE       0
      (12) EISD$V_NONSHRADR 1
   section type: EISD$K_NORMAL
   base VBN: 3
```

```

2) image section descriptor (36 bytes)
   byte count: 512
   base virtual address: %X'00020000' (P0 space)
   page fault cluster size: default
   ISD flags:
       (0) EISD$V_GBL      0
       (1) EISD$V_CRF      0
       (2) EISD$V_DZRO     0
       (3) EISD$V_WRT      0
       (4) EISD$V_INITALCOD 0
       (5) EISD$V_BASED    0
       (6) EISD$V_FIXUPVEC 0
       (7) EISD$V_RESIDENT 0
       (8) EISD$V_VECTOR   0
       (9) EISD$V_PROTECT  0
      (10) EISD$V_LASTCLU   1
      (11) EISD$V_EXE       1
      (12) EISD$V_NONSHRADR 0
   section type: EISD$K_NORMAL
   base VBN: 4

```

```

...
ANA/IMAGE FAULT
$
$

```

Hard faults to activate a simple image uninstalled are 9:

```

$
$ r fault
Total faults: 1516
Hard faults: 296
$
$ r fault
Total faults: 1578
Hard faults: 305
$
$ r fault
Total faults: 1640
Hard faults: 314
$
$ r fault
Total faults: 1703
Hard faults: 323
$

```

Hard faults with image installed shared are 8:

```
$ install replace sys$login:fault/share
$
$
$ run fault
Total faults: 531
Hard faults: 76
$
$ run fault

Total faults: 623
Hard faults: 84
$
$ run fault
Total faults: 694
Hard faults: 92
$
```

Hard faults after installing image with shared address data are 6:

```
$ set proc/priv=all
$ install add sys$login:fault/share=addr
$ r fault
Total faults: 2077
Hard faults: 386
$
```

Shared Address Data Dependencies Example

```
$ r fault
Total faults: 2136
Hard faults: 392
$
$ r fault
Total faults: 2223
Hard faults: 398
$
$ r fault
Total faults: 2284
Hard faults: 404
$
$ install del sys$login:fault
```

Program dependent on two shareable images:

```
$ type f1.c
/* Program to display total and hard faults, since process creation.
   Program uses undocumented interfaces which neither supported
   nor recommended.
   Author: Bruce Ellis
*/

#include <stdio.h>
#include <ssdef.h>
#include <phddef.h>
#include <lib$routines.h>

extern      PHD *ctl$gl_phd;
void x(void);

int main(void)
{
/* Display total and hard faults. */
printf("Total faults: %d\n",ctl$gl_phd->phd$l_pageflts);
printf("Hard faults:  %d\n",ctl$gl_phd->phd$l_pgfltio);
x();
return(SS$_NORMAL);
}
```

```
$ type sh1.c
#include <stdio.h>
void y(void);

void x(void)
{
    printf("Address of x is: %p\n",x);
    y();
}
$ type sh2.c
#include <stdio.h>

void y(void)
{
    printf("Address of y is: %p\n",y);
}
$
```

Building and installing image and shareable images **/SHARE :**

```
$ type build_sh.com
$ cc sh2
$ link/share sh2,sys$input/opt
gsmatch=lequal,1,1
symbol_vector=(y=procedure)
$
$ define/system sh2 dka200:[bae]sh2
$ cc sh1
$ link/share sh1,sys$input/opt
gsmatch=lequal,1,1
symbol_vector=(x=procedure)
sh2/share
$ define/system sh1 dka200:[bae]sh1
$
$ cc f1+sys$library:sys$lib_c/lib
$ link/sysexe f1,sys$input/opt
sh1/share
$
$ @build_sh
%DCL-I-SUPERSEDE, previous value of SH2 has been superseded
%DCL-I-SUPERSEDE, previous value of SH1 has been superseded
$
$ inst add sh1/share
$ inst add sh2/share
$ inst add sys$login:f1/share
$
```

**Note**

Shareable images are mapped into P0 space; 12 hard faults to activate image.

```
$ run f1
Total faults: 3312
Hard faults: 572
Address of x is: 32000
Address of y is: 64000
$ run f1
Total faults: 3411
Hard faults: 588
Address of x is: 32000
Address of y is: 64000
$ run f1
Total faults: 3488
Hard faults: 600
Address of x is: 32000
Address of y is: 64000
$ run f1
Total faults: 3560
Hard faults: 612
Address of x is: 32000
Address of y is: 64000
$
```

**Note**

Because sh1 is dependent on sh2, sh2 must be installed with shared address data before sh1.

```
$ inst repl sh1/share=addr
%INSTALL-I-NOTSHRADR, SH2 is not installed with shareable address data
%INSTALL-I-NONSHRADR, image installed ignoring '/SHARE=ADDRESS'
DISK$AXP71FT2:<BAE>SH1.EXE
$
$
$ inst repl sh2/share=addr
$ inst repl sh1/share=addr
$ inst repl sys$login:f1/share=addr
$
```



Note

After all are installed with shared data, the number of hard faults drops to six per run. Shareable images are mapped to P1 space.

```
$ run f1
Total faults: 4142
Hard faults: 706
Address of x is: 7C944000
Address of y is: 7C912000
$ run f1
Total faults: 4257
Hard faults: 716
Address of x is: 7C944000
Address of y is: 7C912000
$ run f1
Total faults: 4373
Hard faults: 722
Address of x is: 7C944000
Address of y is: 7C912000
$
```

Deinstalling sh2 voids the shared address data at run time for sh1 and f1.



Note

Routines are back in P0 space and hard faults are now 38!

```
$ inst del sh2
$ run f1
Total faults: 4607
Hard faults: 773
Address of x is: 38E000
Address of y is: 3C0000
$ run f1
Total faults: 4675
Hard faults: 811
Address of x is: 38E000
Address of y is: 3C0000
$
```


Monitoring Image Region Pages

```
$ install list/image/full
```

```
P1 Image Region 7B13E000:7FF5DFFF, 10000 pages
```

```
Available Image Region ranges  
7C928000:7FF5DFFF, 6939 pages
```

```
Total of 6939 free pages in 1 blocks. Largest block 6939 pages.  
$
```

Shared and Other Install Options

- **No qualifiers**
 - If an image is installed with no qualifiers, the file ID of the image file is stored in a structure in paged pool.
 - With the file ID, you can open image file without having to perform any I/O operations to do a directory lookup, yielding faster image activation.
- **/OPEN**
 - When an image is installed with **/OPEN** qualifier, the file is kept open.
 - No I/Os are required to open the image file, yielding faster image activation.
 - Cost for installing image **/OPEN** is about 300 bytes of nonpaged pool for data structures required to describe the open image file.
 - Qualifiers **/SHARE** and **/HEADER_RESIDENT** imply **/OPEN**.
- **/HEADER RESIDENT**
 - Causes the image header to be stored in the paged pool.
 - Image header contains the structures required to build memory management data structures at image activation.
 - By storing the header in memory, you can activate image without I/O to read it.
 - Allows for faster image activation, at a cost of roughly one page of paged pool.

Modified Page List and Modified Page Writing

Module 8

Module Outline

Objectives

This module describes the modified page list, its uses, and performance characteristics. Although modified page writing should have minimal impact on performance, it is discussed for completeness.

Topics

- Modified Page List
- Modified Page List Parameters and Metrics
- Modified Page List Performance Considerations
- MPL Performance and Analysis

Modified Page List

Modified Page List Functions

Provides form of cache for data pages incidentally displaced from working set lists.

- Modified pages are data pages that have changed or are likely to have changed, since made valid and have no representation on disk.
- If a process displaces one of these pages due to nature or working set list replacement and needs a page again in a short time, it may soft fault a page from this list.
- No I/O is required to write a page to disk with this activity.
- The modified page writer (**SWAPPER** process) gathers pages owned by many different processes into a single large I/O when the list needs to be written.
- Prevents overhead of many small I/Os.

Modified Page List Parameters

MPW_HILIMIT

- Defines the number of pages allowed on the modified page list prior to waking **SWAPPER** to the write list.
- Prevents depletion of the free page list due to a large modified page list.
- Allows for more regular writing of smaller numbers of pages, instead of a very large amount of activity all at once, due to a list size that has gotten out of control.

MPW_LOLIMIT

- Defines the lower bound of a list, at which point **SWAPPER** will stop performing writes when awakened because the free list is below **FREELIM**.
- Serves two purposes:
 - Guarantees that some pages on the list are not written and stand a good chance of being faulted back into the working set by processes.
 - Allows the size of **MPW_HILIMIT** to be raised without performing a large number of I/Os to write the list.
- Note that some of the traditional functionality attributed to the parameter **MPW_LOLIMIT** has been moved over to the parameter **MPW_LOWAITLIMIT**, such that **MPW_LOLIMIT** is only used when the free page list size has fallen below the parameter **FREELIM**.

Modified Page List Parameters and Metrics

Modified Page List Parameters

MPW_WRTCLUSTER

- Defines the number of pages to write on a single I/O operation.
- Used to preallocate the I/O map from the nonpaged pool.
- Value is typically 96 to 120 to minimize I/O overhead associated with initiating write operation.

MPW_WAITLIMIT

- Defines the absolute upper bound to the size of the modified page list.
- If process attempts to place page on the list and its size has exceeded this value, process is placed in the **RWMPB** flavor of **MWAIT** state.
- Process remains in this state until the list has been written down to **MPW_LOWAITLIMIT** pages in size and then is released from the state.
- **MPW_WAITLIMIT** should never be set to a value less than **MPW_HILIMIT**.

MPW_THRESH

- If the free page list falls below **FREELIM** pages in size, **SWAPPER** wakens to reclaim memory for the free list.
- One check **SWAPPER** performs during reclamation process is to determine if the modified page list has **MPW_THRESH** pages or greater on it.
- If it does have that many pages, **SWAPPER** writes a modified page list in attempt to reclaim memory.

MPW_IOLIMIT

- Defines the number of concurrent I/O operations that **SWAPPER** may perform on page files.
- Allows for multithreaded writes, which allows **SWAPPER** to perform work while I/Os are in progress.
- May also allow for better load balancing of I/O operations to spindles when the system supports multiple page files on different disk drives.

Modified Page Writing Metrics

- **PAGE WRITE I/O RATE** in **\$MONITOR PAGE** identifies the number of times pages are being written from list to page file per second.
- **PAGE WRITE RATE** in **\$MONITOR PAGE** identifies the number of pages written to page file per second.
 - Dividing **PAGE WRITE RATE** by **PAGE WRITE I/O RATE** identifies the level of page write clustering that is being performed.
 - If the value is close to the setting for **MPW_WRTCLUSTER**, the system is performing efficient write operations.
- **WRITE-IN-PROGRESS FAULT RATE** identifies the number of faults being resolved off the **SWAPPER**'s I/O map.
 - Pages were being written to the page file when faulted off the map.
 - Indicates inefficient modified page writing.

Modified Page List Performance Considerations

Parameter Sizing

Raising the **MPW_HI LIMIT** parameter allows more processes to have pages on the modified page list before forcing a write operation.

Modified List Faults and Working Set List Sizes

Although **MODIFIED PAGE LIST** faults are soft faults, remember that faults due to the inability for processes to keep pages in the working set.

- A high modified page list fault rate may indicate the need to increase working set extents and possibly quotas, if the free list is sufficiently robust.
- Keeping pages in the working set will eliminate the need for soft faults and minimize modified page list writing.

MPL Performance and Analysis

Multiple Page Files

- Other than conservation of space, the primary reason to consider more than one page file is high I/O activity due to memory-tight system.
 - If the **WRITE I/O** rate is high (consistently 1 or more), the **READ I/O** rate is high (consistently 5 to 10 or more), and the free list is small, consider multiple page files on separate disks.
 - For more than four, increase parameter **PAGFILCNT**.

Inefficient Modified Page Writing

- If inefficient modified page list writing is occurring, it may be caused by page file fragmentation.
 - The general cure for this problem is a larger page file.
 - If you are swapping process headers and there are plenty of pages on free page list, increase **BALSETCNT**.

Indicators of Modified List Problems

- If processes noticeably show up in **RWMPB**, the modified page list is becoming full enough to stall the processes.
 - Consider a larger setting for **MPW_HILIMIT** and **MPW_WAITLIMIT**.
 - Consider larger working sets.
 - Consider adding additional page files on separate disks (for more optimal I/O).
 - Alternatively, consider a larger page file if it appears to be close to full (prevents temporary exhaustion and minimizes internal fragmentation of page file space).

Page File Exhaustion

Page file space should be monitored to make sure that you are not running out of space.

Page file exhaustion can cause processes to stall in **RWMPB** or **RWPFF** states. The hangs may only be temporary.

To determine how much space is available in the page files use:

\$ **show memory/file/full**

System Memory Resources on 7-DEC-1998 04:32:13.40

DISK\$AXP71FT2:[SYS0.SYSEXE]SWAPFILE.SYS

Free Blocks	20480	Reservable Blocks	20480
Total Size (blocks)	20480	Paging File Number	1
Swap Usage (processes)	0	Paging Usage (processes)	0

This file is used exclusively for swapping.

DISK\$AXP71FT2:[SYS0.SYSEXE]PAGEFILE.SYS

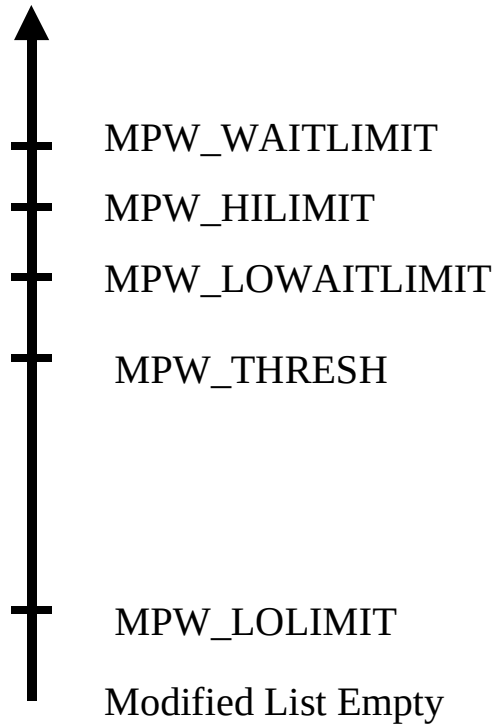
Free Blocks	290816	Reservable Blocks	187888
Total Size (blocks)	290816	Paging File Number	3
Swap Usage (processes)	0	Paging Usage (processes)	29

This file can be used for either paging or swapping.

\$

To install an additional page file, use the **SYSGEN** commands **CREATE** and **INSTALL/PAGEFILE**.

Modified Page Writer Parameter Relationships



The Swapper and Memory Reclamation

Module 9

Module Outline

Objectives

This module discusses the swapper's role in memory reclamation. It also discusses parameters involved in proactive memory reclamation.

Topics

- The Swapper and Memory Reclamation
- Memory Reclamation and Virtual I/O Cache
- Free Page List Parameter Relationships

The Swapper and Memory Reclamation

- The **SWAPPER** reclaims memory when the free list gets small.
When the free list falls to **SYSGEN** parameter **FREELIM** pages in size or less, **SWAPPER** will be awakened.
- If the free list is exhausted when a process generates a fault, the process will be forced into **FPG** wait state.
Processes showing up in this state are an indication that parameter **FREELIM** is too small.
- When the **SWAPPER** is awakened, it performs the following actions until the free page list size has **FREEGOAL** pages or more on it.
SWAPPER checks the list against **FREEGOAL** while performing its operations, and so should stop reclaiming quickly after the free list has attained **FREEGOAL** pages.

Activities for Memory Reclamation

- If the free list is at **MPW_THRESH** pages in size, write it to reclaim pages.
- Reclaim loans (trim process working set lists toward their **WSQUOTA**) based on the likelihood of them running again in the near future.
- Attack processes in categories based on state (and possibly length of time in state), priority, and whether they have direct or buffered I/O in progress.
 - One category will be attacked before going to the next category.
 - Processes in response time categories (**CEF**, **LEF**, and **HIB** with no direct I/O in progress and **SUSP** states with buffered I/O in progress) will be the secondary trimmed within the class prior to swapping in the class.
 - Processes in other classes will be swapped with no secondary trimming.
 - Secondary trimming implies that working set lists will be trimmed to **SYSGEN** parameter **SWPOUTPGCNT** pages in size.



Note

If working set quotas are assigned properly, none of this will be required, other than first-level trimming.

- With Version 5.4-3 of OpenVMS, support was included for proactive memory reclamation. This characteristic is enabled by the setting of flags in the parameter **MMG_CTLFLAGS**.
 - **Bit 0**

If set, this flag enables the shrinking of working sets (without affecting the working set list size) for processes that run periodically but infrequently. This includes processes that have been idle for 30 seconds and that have been idle for 99% of their life.

This behavior occurs when the free page list size falls below **2*FREEGOAL** pages in size.

This process is referred to as the “ticker.”
 - **Bit 1**

If set, this flag enables the swapping of processes that have been idle for **LONGWAIT** (30) seconds. The swap of **SWPOUTPGCNT** pages, but the working set list size is unaffected. An effort is made to keep the process header memory resident.

This behavior occurs when the free page list falls below **FREEGOAL** pages in size.

This process is referred to as the “troller.”

Proactive Memory Reclamation Example

```

$
$ show memory/physical
      System Memory Resources on 25-OCT-1999 13:21:16.96
Physical Memory Usage (pages):      Total      Free      In Use      Modified
Main Memory (8.00Mb)                16384      3782      12197      405
Of the physical pages in use, 4125 pages are permanently allocated to VMS.
$
$ show system
VAX/VMS V5.5-2 on node VASTEK 25-OCT-1999 13:21:27.45 Uptime 0 02:13:22
  Pid   Process Name   State Pri   I/O      CPU      Page flts Ph.Mem
00000021 SWAPPER      HIB   16     0  0 00:01:13.90      0      0
00000024 IPCACP       HIB   10    10  0 00:00:00.09     106     64
00000026 ERRFMT      HIB    8    92  0 00:00:00.58     136     67
00000027 OPCOM       HIB    8    63  0 00:00:01.33     297     79
00000028 AUDIT_SERVER HIB   10    45  0 00:00:00.90    1862     43
00000029 JOB_CONTROL  HIB   10    47  0 00:00:00.39     260     43
0000002A QUEUE_MANAGER HIB    8   183  0 00:00:01.73     566    648
0000002B TP_SERVER    HIB   10   547  0 00:00:06.43    1216    115
0000002C SYMBIONT_1   HIB    4    18  0 00:00:00.30     181    122
0000002D NETACP      HIB    9    51  0 00:00:00.62     307    135
0000002E EVL        HIB    6    57  0 00:00:00.63    2479     34 N
0000002F REMACP      HIB    9    17  0 00:00:00.11      82     42
00000031 DECW$SERVER_0    HIB    7   312  0 00:00:15.91    3798    442
00000032 ELLIS       LEF    4    95  0 00:00:03.23    3368   1264
00000033 DECW$WM_1     LEF    7    45  0 00:00:02.16    1307   1490
00000035 DECW$TE_1     COM    4   3099  0 00:02:45.93    2484   1472
00000036 _TWA2:      HIB    7   2703  0 00:00:26.13    6862    238
00000038 NETBIOS     HIB    9    30  0 00:00:00.31     141    181
0000003B PCFS_SERVER HIB   10   665  0 00:00:06.33    4366   2023
0000003C _RTA1:     CUR    4   120  0 00:00:01.23     479    439
$
$ mcr sysgen
SYSGEN> SHOW FREEGOAL
Parameter Name      Current   Default   Min.      Max.      Unit      Dynamic
-----
FREEGOAL            2048     200       16        -1 Pages   D
SYSGEN> SET FREEGOAL 4000
SYSGEN> WRITE ACTIVE
SYSGEN>
SYSGEN> EXIT
$

```

\$ show system

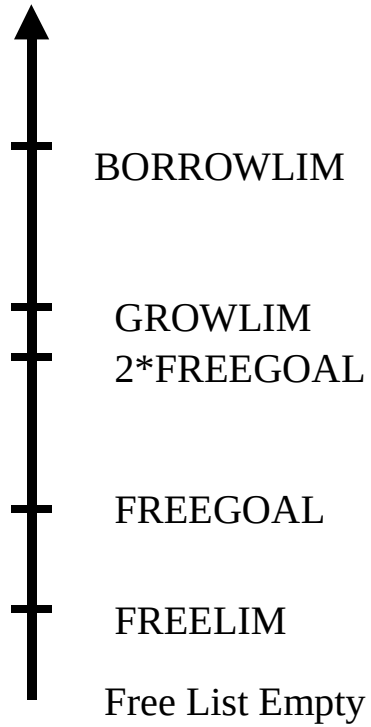
```
VAX/VMS V5.5-2 on node VASTK 25-OCT-1999 13:23:33.05 Uptime 0 02:15:28
  Pid   Process Name   State Pri    I/O      CPU      Page flts Ph.Mem
00000021 SWAPPER        HIB   16     0    0 00:01:13.92      0      0
00000024 IPCACP          HIB   10    10    0 00:00:00.09     106     64
00000026 ERRFMT        HIB    8    92    0 00:00:00.58     136     67
00000027 OPCOM         HIB    6    65    0 00:00:01.43     337    110
00000028 AUDIT_SERVER   HIB   10    45    0 00:00:00.90    1903     41
00000029 JOB_CONTROL     HIB   10    47    0 00:00:00.39     262     31
0000002A QUEUE_MANAGER HIB    8   183    0 00:00:01.73     566    648
0000002B TP_SERVER       HIB   10   555    0 00:00:06.58    1341    122
0000002C SYMBIONT_1    HIBO   4     --  swapped out  --      107
0000002D NETACP        HIB   10    51    0 00:00:00.62     333    128
0000002E EVL           HIB    6    57    0 00:00:00.63    2552     32 N
0000002F REMACP        HIB    9    17    0 00:00:00.11      82     42
00000031 DECW$SERVER_0      HIB    6   312    0 00:00:17.68    3937    378
00000032 ELLIS            LEF    4    95    0 00:00:03.23    3368   1264
00000033 DECW$WM_1        LEF    7    45    0 00:00:02.16    1307   1490
00000035 DECW$TE_1        COM    4   3506   0 00:02:50.30    2484   1472
00000036 _TWA2:           HIB    7   2783   0 00:00:26.21    6879    114
00000038 NETBIOS         HIBO   9     --  swapped out  --      171
0000003B PCFS_SERVER    HIB   10   674    0 00:00:06.46    4714   2023
0000003C _RTA1:         CUR    4    190    0 00:00:02.11     741    262
$
```

Memory Reclamation and Virtual I/O Cache

Interaction Between Virtual I/O and Proactive Memory Reclamation

- Cache can grow up to **VCC_MAXSIZE** (up to **VCC_PTES** on **VAX**) if:
 - Free List size $\geq 2 \times \mathbf{FREEGOAL}$ and ticker is on.
 - Free List size $\geq \mathbf{FREEGOAL}$ and troller is on.
 - Free List size $> \mathbf{GROWLIM}$ and ticker and troller are off.
- Memory management subsystem can force returning pages, trims using first-level trimming; then returns in small increments.

Free Page List Parameter Relationships



Module Outline

Objectives

This module discusses the cost and impact of system faulting.

Topics

- System Faults

System Faults

The Nature of System Faults

Portions of system space are pageable.

- All pageable system space, with the exception of balance slot area (processes are charged for their own process headers), is charged against the system's working set list.
- The system's working set list is sized by **SYSGEN** parameter **SYSMWCNT**.
- If too small, system faults will occur and can be traced in **\$MONITOR PAGE**.
- If too large, pageable portions of system space that are rarely used could tie up physical memory unnecessarily.

How Many System Faults Are Too Many?

Remember that these faults are processed the same way as other faults.

- If there are 5 or 10 system per second faults that are soft, the impact on the system will be minimal.
- This is not critical. You will tend to find that if the system fault rate is high, it will be fairly high, in the 10s to 100s.
- As **SYSMWCNT** increases, the faults will reach a threshold where they tend to disappear.
- Again, a few system faults per second is usually not critical, assuming they are soft faults.

Module Outline

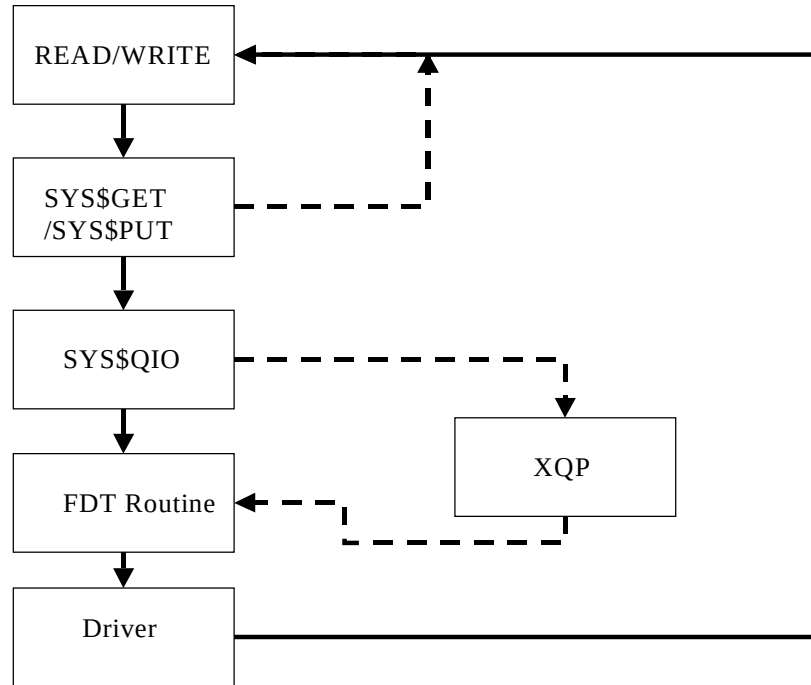
Objectives

This module describes the fundamental characteristics of OpenVMS I/O. The module describes I/O from a high-level language READ/WRITE down to the basics of the hardware implementation of I/O. The concentration in this module will be basic performance impacts of the various components of I/O.

Topics

- General I/O Flow
- Geometry of Disks
- Time-Based Components of a Disk Transfer
- Optimization of Disk Transfers
- Virtual I/O Cache

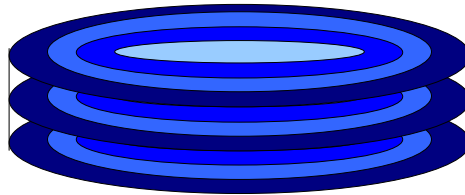
General I/O Flow



- When a user program issues file input/output (I/O) such as **READ** or **WRITE**, the request gets translated into a call to record management services (RMS) request to **SYS\$GET** (read) or **SYS\$PUT** (write) a record.
- RMS buffers I/O requests, depending on the type of file organization. Buffering means the actual I/O operation may not be required.
- If a record was not contained in the local buffer for read operation or if the buffer is full on write operation, RMS translates the request to system service **SYS\$QIO**.
- **SYS\$QIO** system service receives
 - Starting virtual block of transfer
 - Size in bytes of transfer
 - Location of where to store data transferred in memory
- **SYS\$QIO** system service passes a request to the Function Decision Table (FDT) routines.

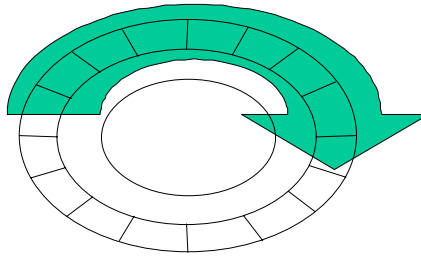
- FDT routines support any preprocessing required on a disk request.
 - Preprocessing includes (not limited to) translation of the starting virtual block in file to the logical block on disk.
 - Translation is done by examining the window control block (**WCB**) data structure.
 - **WCB** contains mapping pointers describing the location of logically contiguous blocks on disk corresponding to virtual block from file.
 - If the file is not logically contiguous on disk (or fragmented), **WCB** can contain many mapping pointers.
 - If the I/O request can be successfully translated, it is passed to the driver.
- **WCB** is allocated from nonpageable system memory and is limited in size so that it will not use too much system memory.
 - Holds a limited number of mapping pointers.
 - If virtual blocks of interest are not described by **WCB**, FDT routines must transfer control to extended qio procedures (**XQP**) to remap "window" to point to blocks of interest.
 - Transfer to **XQP** to remap **WCB** is called window turn.
 - Window is turned on by **XQP** and the I/O request continues.
- Driver eventually receives the request and sets up a disk drive to support the transfer to or from disk.
 - Waits for interrupt to signal completion of the transfer
 - When interrupt is received, potential errors processed
 - If there are no errors, driver requests completion
- Driver requests completion, but it may not actually be done with the request.
 - If transfer is virtual, not logically contiguous, only first part of transfer is completed.
 - Disk controller can only handle logically contiguous requests.
 - First part of the transfer has been completed.
 - If more blocks associated with the transfer are different logical blocks, the I/O request must be remapped and reinitiated.
 - Operation is referred to as a split transfer.
 - On very fragmented files, there can be many split transfers.

Geometry of Disks



Cylinders

Sectors and Disk
Rotation



- Files are organized by virtual blocks.
 - All files start at VBN 1 and grow to VBN x, where x is the size of file in blocks.
 - Each virtual block in a file corresponds to the logical block on disk.
 - Disks start at logical block number (LBN) 0 and grow to LBN x-1, where x is the size of disk in blocks.
 - A set of logically contiguous blocks allocated to the same file is called an extent.
- Disk drive contains read/write heads placed on servo.
 - Servo looks like a comb with read/write heads on end and sweeps across the surfaces of disk.
 - Disk is organized physically in sectors, tracks, and cylinders.
 - Disk is made up of platters separated on a spindle.
 - Platters are broken up into concentric circles called tracks.
 - Tracks are broken up into 512-byte sectors corresponding to disk blocks.
 - If one drew a perpendicular line through a track, all tracks intersected would form a cylinder.
- Component of disk transfer is performing seek operation or positioning read/write heads on servo over proper cylinder boundary.
 - Disk spins in the drive, traditionally at 3600/7200 RPM (60/120 revolutions per second).
 - Another component of disk transfer waits for the starting block of transfer to rotate under read/write heads.
 - Wait time is referred to as rotational latency.

Time-Based Components of a Disk Transfer

- Initiation
 - First portion of disk transfer is made up of the central processing unit (CPU) overhead of **READ/WRITE**, **SYSS\$GET/SYSS\$PUT**, and **SYSS\$QIO** call.
 - This time is CPU bound.
 - Time is CPU bound and becomes insignificant as you move to faster VAX and Alpha processors.
- Seek
 - May not be necessary if heads are on cylinder.
 - In most situations, seeks are necessary.
 - Mechanical component of disk transfer and fundamentally slowest component.
 - Minimizing, eliminating, and/or optimizing seek operations is the key to improving disk performance.
 - Faster CPU will not improve this component of I/O and can cause degradation due to more demand on drive.
- Rotational Latency
 - Wait time for the starting block to rotate under read/write heads is the next slowest component of disk transfer.
 - Bound by the speed of disk rotation.
 - Can be optimized by proper access to drive.
- Transfer Time
 - Bounded by wire speed.
 - When the drive is transferring on the controller, it is blocking other transfers by drives on that disk controller.
 - Spreading I/Os across multiple controllers can minimize waits to initiate transfer.
- Operation Completion
 - Transfer is CPU bound and consists of processing completion interrupt and queuing AST to process to signal I/O completion.
 - Little can be done to improve time.

Optimization of Disk Operations

- Seeking (followed by rotational latency) is the most expensive component of disk transfers.
 - Improving these would provide a logical starting point in improving the performance of disk transfers.
 - These are the easiest components to control.
- Contiguity Considerations
 - If files are logically contiguous on disk and transfers to files are sufficiently large, read/write heads have to move less, and there is less rotational latency overhead.
 - Adjacent blocks processed while disk spinning.
 - Making files more contiguous and performing large transfers can improve the performance characteristics of large transfers.
- Spreading I/O Operations Across Disks
- Drives' read/write heads can move in parallel.
 - Helps minimize contention for a single resource.
 - Allows multiple operations to be performed simultaneously - improving throughput and response time of disk I/O operations.
- Several hardware and software components enhance file-related performance characteristics, including:
 - Controller seek optimizations
 - Shadowing
 - Striping
 - Virtual I/O Cache
 - RAID

Virtual I/O Cache

- Virtual I/O cache is available to improve disk I/O performance.
- Pages are taken from the free page list, up to **VCC_MAXSIZE** (limited by **VCC_PTES** on VAX).
- Should be considered when sizing working set sizes.
- May be disabled by setting **VCC_FLAGS** to 0 (**VBN_CACHE_S** on VAX).
- Must be enabled on ALL nodes in a OpenVMS Cluster to be in effect.
- File-oriented (not block-oriented) cache.

```
$ mcr sysgen
SYSGEN>
SYSGEN> SHOW VCC_FLAGS
Parameter Name      Current      Default      Min.      Max.      Unit
Dynamic
-----
-
VCC_FLAGS            1            1            0         -1 Bitmask
SYSGEN> SHOW VCC_MAXSIZE
Parameter Name      Current      Default      Min.      Max.      Unit
Dynamic
-----
-
VCC_MAXSIZE          6400         6400         02000000000 Blocks
SYSGEN>
SYSGEN>
SYSGEN>
$
$
$ show memory/cache/full
                        System Memory Resources on 1-SEP-1999 15:47:36.26
Virtual I/O Cache
  Total Size (Kbytes)      3200      Read IO Count      46792
  Free Kbytes              0         Read Hit Count     42430
  Kbytes in Use            3200      Read Hit Rate       90%
  Write IO Bypassing Cache 0         Write IO Count      201
  Files Retained           92        Read IO Bypassing Cache 1171
$
```

Analyzing the Effects of Fragmentation

Module 12

Module Outline

Objectives

This module illustrates the effect of minimizing fragmentation and maximizing spindle utilization on disk performance.

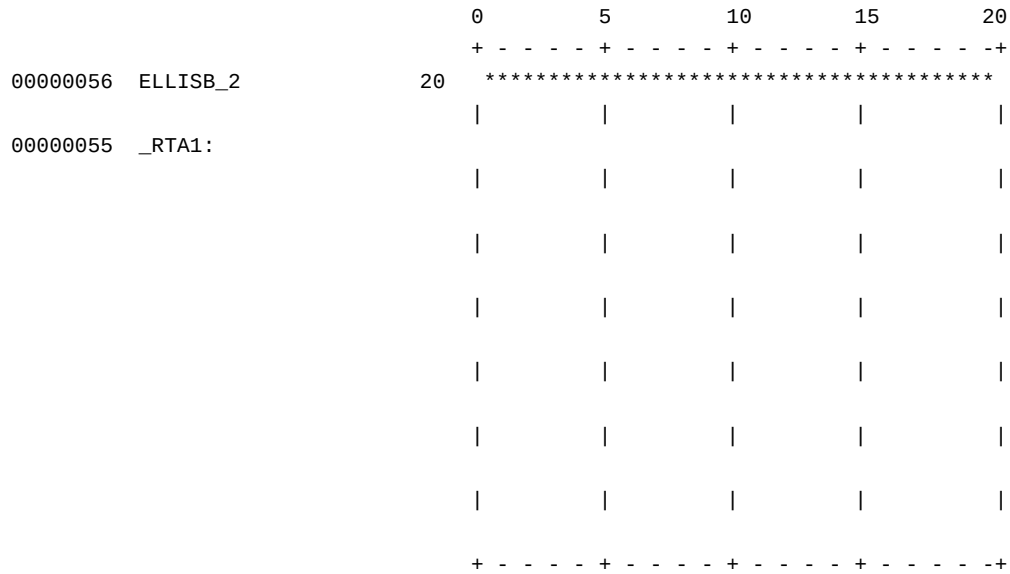
Topics

- Determining I/O Rates
- Dump of Header for X.Dat
- Effect of Making File Contiguous
- Effect of Making Both Files Contiguous
- Using Multiple Spindles

Determining I/O Rates

\$ MONITOR DISK/ITEM=ALL, PROCESS/TOPDIO

VAX/VMS Monitor Utility
TOP DIRECT I/O RATE PROCESSES
on node BRUDEN
8-NOV-1999 22:55:24



VAX/VMS Monitor Utility
DISK I/O STATISTICS
on node BRUDEN
8-NOV-1999 22:55:24

I/O Operation Rate		CUR	AVE	MIN	MAX
\$1\$DUA0:	(BRUDEN) LABSYSRL5	0.00	0.00	0.00	0.00
\$1\$DUB1:	(BRUDEN) BRUDEN	32.05	33.49	32.05	34.92

VAX/VMS Monitor Utility
DISK I/O STATISTICS
on node BRUDEN
25-OCT-1992 14:10:28

I/O Request Queue Length		CUR	AVE	MIN	MAX
\$1\$DUA0:	(BRUDEN) LABSYSRL5	0.00	0.00	0.00	0.00
\$1\$DUB1:	(BRUDEN) BRUDEN	0.96	0.96	0.95	0.96

- In this display, through **\$MONITOR DISK/ITEM=ALL, PROCESS/TOPDIO**, process **ELLISB_2** is generating 20 direct I/O operations per second.
 - Disk **DUB1:** has roughly 33 I/O operations performed on it.
 - No other disks have any traffic.
 - Implies that the **ELLISB_2** process is responsible for I/Os to **DUB1:** and files being processed on the volume.

- To determine files open on the volume, issue:

```
$ SHOW DEVICE/FILES DUB1:
```

```
$ show device/files
```

```
Files accessed on device $1$DUB1: (BRUDEN) on 8-NOV-1999 22:56:17.53
```

Process name	PID	File name
	00000000	no privilege for attempted operation
ELLISB_2	00000056	[USERS.ELLISB.IO_VIDEO]SPLIT.EXE;2
ELLISB	00000046	[USERS.ELLISB.IO_VIDEO]DISK1.LOG;1
ELLISB_2	00000056	[USERS.ELLISB.IO_VIDEO]X.DAT;2
ELLISB_2	00000056	[USERS.ELLISB.IO_VIDEO]Y.DAT;2
\$		

- Aside from program **SPLIT.EXE**, two data files are open: **X.DAT** and **Y.DAT**.
- To determine the cause of the extra 13 I/O operations performed on disk, examine the file header for **X.DAT**.
- In the map area of the file header, there are several retrieval pointers defining locations of extents that make up the file.
- The spread of **LBNs** described by the mapping pointers implies additional seeking and, more commonly, rotational latency overhead.
- The reason for additional I/Os to **DUB1:** is that with this much fragmentation, split transfers are being generated.
- Another point on examining X.DAT is that the extension file identification is nonzero.
 - This implies that the extension file header contains information that could not fit in the first header block.
 - The information in the extension header is additional mapping/retrieval pointers.

Dump of Header for X.Dat

\$ DUMP/HEADER/BLOCK=COUNT:0 X.DAT

Dump of file DUB1:[USERS.ELLISB.IO_VIDEO]X.DAT;2 on 8-NOV-1999
22:56:42.96

File ID (5235,5,0) End of file block 5000 / Allocated 5001

File Header

Header area

Identification area offset:	40
Map area offset:	100
Access control area offset:	255
Reserved area offset:	255
Extension segment number:	0
Structure level and version:	2, 1
File identification:	(5235,5,0)
Extension file identification:	(5239,5,0)
VAX-11 RMS attributes	
Record type:	Fixed
File organization:	Sequential
Record attributes:	<none specified>
Record size:	512
Highest block:	5001
End of file block:	5001
End of file byte:	0
Bucket size:	0
Fixed control area size:	0
Maximum record size:	512
Default extension size:	0
Global buffer count:	0
Directory version limit:	0
File characteristics:	<none specified>
Map area words in use:	154
Access mode:	0
File owner UIC:	[VPRB,ELLISB]
File protection:	S:RWED, O:RWED, G:RE, W:
Back link file identification:	(2873,12,0)
Journal control flags:	<none specified>
Active recovery units:	None
Highest block written:	5001

Identification area

File name: X.DAT;2
 Revision number: 1
 Creation date: 8-NOV-1999 22:50:32.60
 Revision date: 8-NOV-1999 22:51:24.49
 Expiration date: 24-JAN-2000 22:51:24.49
 Backup date: <none specified>

Map area

Retrieval pointers

Count:	3	LBN:	489735
Count:	12	LBN:	489741
Count:	6	LBN:	495483
Count:	6	LBN:	502881
Count:	6	LBN:	504660
Count:	3	LBN:	505818
Count:	9	LBN:	513777
Count:	9	LBN:	526329
Count:	15	LBN:	526503
Count:	33	LBN:	526551
Count:	12	LBN:	527331
Count:	3	LBN:	535557

Dump of file DUB1:[USERS.ELLISB.IO_VIDEO]X.DAT;2 on 8-NOV-1999
 22:56:42.96

File ID (5235,5,0) End of file block 5000 / Allocated 5001

Count:	3	LBN:	542592
Count:	6	LBN:	546810
Count:	9	LBN:	552252
Count:	3	LBN:	552414
Count:	30	LBN:	564087
Count:	27	LBN:	592572
Count:	3	LBN:	593286
Count:	3	LBN:	593451
Count:	6	LBN:	602145
Count:	6	LBN:	603471
Count:	9	LBN:	612942
Count:	3	LBN:	614043
Count:	3	LBN:	614181
Count:	15	LBN:	655008
Count:	3	LBN:	655173
Count:	15	LBN:	249834

Count :	33	LBN :	249882
Count :	33	LBN :	249948
Count :	33	LBN :	250014
Count :	33	LBN :	250080
Count :	33	LBN :	250146
Count :	33	LBN :	250212
Count :	33	LBN :	250278
Count :	33	LBN :	250344
Count :	33	LBN :	250410
Count :	33	LBN :	250476
Count :	33	LBN :	250542
Count :	33	LBN :	250608
Count :	33	LBN :	250674
Count :	33	LBN :	250740
Count :	33	LBN :	250806
Count :	33	LBN :	250872
Count :	33	LBN :	250938
Count :	33	LBN :	251004
Count :	33	LBN :	251070
Count :	33	LBN :	251136
Count :	33	LBN :	251202
Count :	33	LBN :	251268
Count :	33	LBN :	251334
Count :	33	LBN :	251400
Count :	33	LBN :	251850
Count :	33	LBN :	251916
Count :	33	LBN :	251982
Count :	33	LBN :	252048
Count :	33	LBN :	252114
Count :	33	LBN :	252180
Count :	33	LBN :	252246
Count :	33	LBN :	252312
Count :	33	LBN :	252378
Count :	33	LBN :	252444
Count :	33	LBN :	252510
Count :	33	LBN :	252576
Count :	33	LBN :	252642
Count :	33	LBN :	252708
Count :	33	LBN :	252774
Count :	33	LBN :	252840

Dump of file DUB1:[USERS.ELLISB.IO_VIDEO]X.DAT;2 on 8-NOV-1999

22:56:42.96

File ID (5235,5,0) End of file block 5000 / Allocated 5001

Count:	33	LBN:	252906
Count:	33	LBN:	252972
Count:	33	LBN:	253038
Count:	33	LBN:	253104
Count:	33	LBN:	253170
Count:	33	LBN:	253236
Count:	33	LBN:	253302
Count:	33	LBN:	253368

Checksum: 40140

Dump of file DUB1:[USERS.ELLISB.IO_VIDEO]X.DAT;2 on 8-NOV-1999
22:56:42.96
File ID (5235,5,0) End of file block 5000 / Allocated 5001

File Header

Header area

Identification area offset:	40
Map area offset:	100
Access control area offset:	255
Reserved area offset:	255
Extension segment number:	1
Structure level and version:	2, 1
File identification:	(5239,5,0)
Extension file identification:	(0,0,0)
VAX-11 RMS attributes	
Record type:	Fixed
File organization:	Sequential
Record attributes:	<none specified>
Record size:	512
Highest block:	1851
End of file block:	1
End of file byte:	0
Bucket size:	0
Fixed control area size:	0
Maximum record size:	512
Default extension size:	0
Global buffer count:	0
Directory version limit:	0
File characteristics:	<none specified>
Map area words in use:	192
Access mode:	0
File owner UIC:	[VPRB,ELLISB]
File protection:	S:RWED, O:RWED, G:RE, W:
Back link file identification:	(5235,5,0)
Journal control flags:	<none specified>
Active recovery units:	None
Highest block written:	1851

Identification area

File name:	X.DAT;2
------------	---------

Map area

Retrieval pointers

Count: 33 LBN: 253434

Count: 33 LBN: 253500

Count: 33 LBN: 253566

Count: 33 LBN: 253632

.

.

.

\$

Effect of Making File Contiguous

- To improve performance of the application, we stop the **ELLISB_2** process in the following display.
 - Issue **\$COPY/CONTIGUOUS** command on **X.DAT**.
 - Should minimize some rotational latency, seeking, and split transfers when accessing the file.
 - Dumping the file header for **X.DAT** shows that the file is contiguous - requires only one retrieval pointer to locate blocks for file.
- When the application is restarted, number of direct I/O operations being generated by the **ELLISB_1** process has increased due to less overhead involved in I/Os.
 - Remember, this is the same application generating the same number of total I/O operations from start to finish.
 - Should run about 10 percent faster.
 - Number of I/Os on **DUB1:** has gone down – fewer split transfers (actually none to file **X.DAT**).

```
$ stop ellisb_2
$
$ copy/contiguous x.dat x.dat
$
$ dump/header/block=count:0 x.dat
```

```
Dump of file DUB1:[USERS.ELLISB.IO_VIDEO]X.DAT;3 on 8-NOV-1999
22:58:30.34
File ID (5270,4,0)   End of file block 5000 / Allocated 5001
```

File Header

Header area

Identification area offset:	40
Map area offset:	100
Access control area offset:	255
Reserved area offset:	255
Extension segment number:	0
Structure level and version:	2, 1
File identification:	(5270,4,0)
Extension file identification:	(0,0,0)
Journal control flags:	<none specified>
Active recovery units:	None
Highest block written:	5001

...

Identification area

File name:	X.DAT;3
Revision number:	1
Creation date:	8-NOV-1999 22:58:10.60
Revision date:	8-NOV-1999 22:58:25.34
Expiration date:	24-JAN-2000 22:51:25.34
Backup date:	<none specified>

Map area

Retrieval pointers

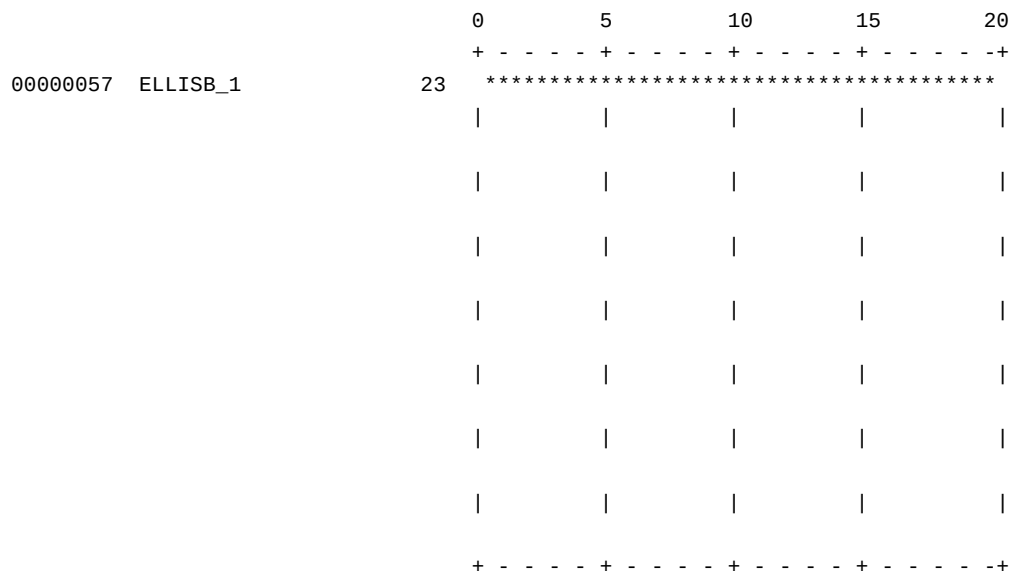
Count:	5001	LBN:	284286
--------	------	------	--------

Checksum:	53134
-----------	-------

\$

\$MONITOR DISK,PROCESS/TOPDIO

```
VAX/VMS Monitor Utility
TOP DIRECT I/O RATE PROCESSES
      on node BRUDEN
      8-NOV-1999 22:59:35
```



```
VAX/VMS Monitor Utility
  DISK I/O STATISTICS
    on node BRUDEN
  8-NOV-1999 22:59:38
```

I/O	Operation	Rate	CUR	AVE	MIN	MAX
\$1\$DUA0:	(BRUDEN)	LABSYSRL5	0.00	0.06	0.00	0.32
\$1\$DUB1:	(BRUDEN)	BRUDEN	28.19	28.44	28.01	29.08

Effect of Making Both Files Contiguous

- In the next display, file **Y.DAT** is contiguous.
 - In the file header dump, not only is the file contiguous, but it is also relatively close to file **X.DAT**.
 - Restarting the application now shows 27 I/Os being generated by **ELLISB_1**, while I/Os to **DUB1**: roughly match the number of direct I/Os being generated by process.
 - Improvement is the result of less I/O overhead and no split transfers being generated.

```
$
$ copy/contiguous y.dat y.dat
$
$ dump/header/block=count:0 y.dat
```

```
Dump of file DUB1:[USERS.ELLISB.IO_VIDEO]Y.DAT;4 on 8-NOV-1999
23:01:21.86
File ID (5273,4,0)   End of file block 2944 / Allocated 5001
```

File Header

Header area

Identification area offset:	40
Map area offset:	100
Access control area offset:	255
Reserved area offset:	255
Extension segment number:	0
Structure level and version:	2, 1
File identification:	(5273,4,0)
Extension file identification:	(0,0,0)
VAX-11 RMS attributes	
Record type:	Fixed

...

File characteristics:	Contiguous
Map area words in use:	3
Access mode:	0
File owner UIC:	[VPRB,ELLISB]
File protection:	S:RWED, O:RWED, G:RE, W:
Back link file identification:	(2873,12,0)
Journal control flags:	<none specified>
Active recovery units:	None
Highest block written:	5001
Identification area	
File name:	Y.DAT;4
Revision number:	1
Creation date:	8-NOV-1999 23:01:07.77
Revision date:	8-NOV-1999 23:01:14.81
Expiration date:	24-JAN-2000 23:01:14.81•
Backup date:	<none specified>
Map area	
Retrieval pointers	
Count: 5001	LBN: 319380
Checksum:	4926
\$	

\$ MONITOR DISK,PROCESS/TOPDIO

VAX/VMS Monitor Utility
 TOP DIRECT I/O RATE PROCESSES
 on node BRUDEN
 8-NOV-1999 23:03:30

		0	5	10	15	20												
		+	-	-	-	+	-	-	-	+	-	-	-	+	-	-	-	+
00000063	ELLISB_1	27	*****															
00000055	_RTA1:																	
						+	-	-	-	+	-	-	-	+	-	-	-	+

VAX/VMS Monitor Utility
 DISK I/O STATISTICS
 on node BRUDEN
 8-NOV-1999 23:03:33

I/O Operation Rate		CUR	AVE	MIN	MAX
\$1\$DUA0:	(BRUDEN) LABSYSRL5	0.00	0.00	0.00	0.00
\$1\$DUB1:	(BRUDEN) BRUDEN	25.88	26.52	25.88	27.24

Using Multiple Spindles

- In the following display, we attempt to improve I/O performance by copying **X.DAT** to separate disk **DUA0: .**
 - Now two sets of read/write heads are active simultaneously.
 - Restarting the application, **ELLISB_1** now shows up to 34 direct I/O operations per second and I/Os are being spread almost evenly between the two disk drives.

```
$ COPY/CONTIGUOUS X.DAT DUA0:[ELLIS]
$ DEFINE/JOB X DUA0:[ELLIS]X.DAT
$ SPAWN/NOWAIT RUN SPLIT
$ MONITOR DISK,PROCESS/TPDIO
```

VAX/VMS Monitor Utility
TOP DIRECT I/O RATE PROCESSES
on node BRUDEN
8-NOV-1999 23:10:40

	0	5	10	15	20
00000071 ELLISB_1	34				
00000055 _RTA1:					

VAX/VMS Monitor Utility
DISK I/O STATISTICS
on node BRUDEN
8-NOV-1999 23:10:40

I/O Operation Rate	CUR	AVE	MIN	MAX
\$1\$DUA0: (BRUDEN) LABSYSRL5	16.66	16.92	16.66	17.17
\$1\$DUB1: (BRUDEN) BRUDEN	16.66	17.08	16.66	17.48

Module Outline

Objectives

This module describes general controller optimizations and their affect on performance.

Topics

- Controller Optimizations
- Locating Bottlenecks
- Other Optimizations Through Layered Products
 - Volume Shadowing
 - Disk Striping
 - RAID

Controller Optimizations

Adapters also support the queue of requests received by the controller in succession.

- Requests are optimized by performing a **SCAN** operation on the queue.
- Commonly referred to as *elevator seek*.
- Requests in the queue are processed based on current movement of read/write heads.
- If heads move toward inner cylinders, the next lowest forward request in the command queue is processed next.
- If heads move toward outer cylinders, the next highest forward request in the command queue is processed next.
- Allows for smooth movement of read/write heads in and then out, as opposed to constant movement to and from random cylinders.

Hierarchical Storage Controllers (HSx)

- **HSC50** was introduced in the middle of Version 3 release.
 - Later versions of **HSC** came out with additional optimizations and support, including **HSC70**, **HSC60**, and **HSC90**. Additional bus support was provided with HSJ, HSD, and HSZ controllers.
 - Controllers are referred to as **HSxs**.
- HSC performs the following seek optimizations, depending on the size of the command queue to disk drive:
 - Zero to two requests in command queue: no optimization.
 - Three to six requests in command queue: **SCAN** optimization.
 - Seven or more requests in command queue: **C-SCAN** optimization. This is similar to **SCAN** except optimizations are always performed in same direction until forward command queue is empty. This allows for equal bias across all cylinders, when there is a large number of requests in queue.
- **HSC** also performs rotational optimizations.
 - If heads are positioned in the middle of transfer, the ending portion of transfer can be processed before the starting portion.
 - This means that a transfer of a track can be performed in no more than one logical rotation of the disk.
- SCSI disks perform **SCAN** flavored optimizations at the driver level.

Example of SCSI optimizations:

```

$
$ type q_stall.mar
    .psect data    noexe,noshr,wrt
    .macro check   ?l
    blbs    r0,l
    $exit_s r0

l:
    .endm  check
d:  .ascid /SYS$DISK:/
c:  .blk1 1
buff: .blkb 512*20
    .psect code    exe,shr,nowrt,page
,*****
;Align on page boundry
,*****
    .align page
    .entry disk_qio,^m<>
    $assign_s      chan=c,devnam=d
    check
,*****
; do 12 asynch $qios with an AST
,*****
    $qio_s efn=#32,chan=c,func=#IO$_READLBLK,astadr = stall,-
        astprm = #10, p1=buff,p2=#20,p3=#10
    $qio_s efn=#33,chan=c,func=#IO$_READLBLK,astadr = stall,-
        astprm = #10000, p1=buff,p2=#20,p3=#10000
    $qio_s efn=#34,chan=c,func=#IO$_READLBLK,astadr = stall,-
        astprm = #100, p1=buff,p2=#20,p3=#100
    $qio_s efn=#35,chan=c,func=#IO$_READLBLK,astadr = stall,-
        astprm = #200, p1=buff,p2=#20,p3=#200
    $qio_s efn=#36,chan=c,func=#IO$_READLBLK,astadr = stall,-
        astprm = #150, p1=buff,p2=#20,p3=#150
    $qio_s efn=#37,chan=c,func=#IO$_READLBLK,astadr = stall,-
        astprm = #20000, p1=buff,p2=#20,p3=#20000
    $qio_s efn=#38,chan=c,func=#IO$_READLBLK,astadr = stall,-
        astprm = #50, p1=buff,p2=#20,p3=#50

```

```
$qio_s      efn=#39,chan=c,func=#IO$_READLBLK,astadr = stall,-
            astprm = #700, p1=buff,p2=#20,p3=#700
    $qio_s   efn=#40,chan=c,func=#IO$_READLBLK,astadr = stall,-
            astprm = #500, p1=buff,p2=#20,p3=#500
    $qio_s   efn=#41,chan=c,func=#IO$_READLBLK,astadr = stall,-
            astprm = #400, p1=buff,p2=#20,p3=#400
$wfland_s   efn=#32,mask=#^x3FF
    check
    ret
    .psect data    noexe,noshr,wrt
buf:
number:      .long  100
    .address   10$
10$: .blkb  100
fmt: .ascid $I/O completing on block: !UL $
    .psect code   exe,shr,nowrt,page
,*****
;Align on a page boundry
,*****
    .align page
    .entry  stall,^m<>
    movl   #100,buf
    $fao_s ctrstr=fmt,outbuf=buf,outlen=buf,p1=<4(ap)>
    check
    pushal number
    calls  #1,g^lib$put_output
    ret
    .end   disk_qio
$ set proc/priv=log_io
$ run q_stall
I/O completing on block: 10
I/O completing on block: 50
I/O completing on block: 100
I/O completing on block: 150
I/O completing on block: 200
I/O completing on block: 400
I/O completing on block: 500
I/O completing on block: 700
I/O completing on block: 10000
I/O completing on block: 20000
$
```


Locating Bottlenecks

OpenVMS provides two fundamental metrics for disk performance: I/O operations per second and disk queue depth.

- "I/O operations per second" is most useful in locating disk drive bottlenecks.
 - Assuming that all transfers are generally small, the disk drive eventually saturates at a given number of I/O operations per second.
 - High I/O rate along with response time degradation, most likely indicates a disk drive bottleneck.
- Queue depth to disk drive defines the length of line to disk drive.
 - The longer the line, the longer the request must wait to be processed.
 - High I/O rate along with high queue depth probably indicates a disk drive bottleneck.
 - High queue depth along with reasonable I/O rate could indicate a bottleneck elsewhere.
 - Other areas could be caused by adapter or controller bottleneck, requestor bottleneck, or many very large transfers.
 - To determine the location of a bottleneck, investigate which disk drives are incurring high queue depths and their physical connections.

Other Optimizatons Through Layered Products

Volume Shadowing

- Allows two or three disks of the same geometry to be treated as shadow set members.
 - Each shadow set member is maintained as a logically identical copy of the other.
 - Intended to improve availability - one member of a shadow set can fail and operations will continue with a short hiccup (transparently) to surviving members of the shadow set.
 - To improve performance of write operations to shadow set members (multiple writes must be performed), members should be placed on separate requestors.
- Members are logically identical - read requests can be processed on any member of the shadow set.
 - Requests are made to a member of the shadow set closest to being on the cylinder.
 - Additional response time can be gained, due to balancing of seek load across members of the shadow set; this keeps requests spread across read/write heads.
 - Read-to-write ratio of 3+ to 1; probably benefits performance from volume shadowing.
 - Typical applications have a 10 to 1 read-to-write ratio.

Disk Striping

- Disk striping allows establishing disk drives as members of the stripe set.
 - When processing the stripe set, a set of logical blocks is located on the first member of stripe set.
 - The next set of logical blocks is located on the next member of the stripe set.
 - Then go either to the next member of the stripe set for next blocks or back to the original member if there are no more members.
 - Allows for some seek load balancing across disk drives, as well as the parallel processing of a large request across members of the stripe set.

RAID

RAID is a Redundant Array of Independent Disks.

- RAID attempts to provide some level of availability and performance at a minimized cost.
- The Industry provides definitions for several levels of RAID, including the following. (Note that definitions also exist for RAID levels 6 (as defined by the RAID Advisory Board) and 7 (as implemented by a third party product)).

RAID Level	Data Distribution	Redundancy	Cost	Performance	Availability
0	Striping	None	Cheapest	Good	Poor
1	Traditional	Shadowing	Expensive	Good for reads/poorer for writes	Excellent
3	Striping	XOR on one disk	Cheaper than shadowing	Good	Good
5	Striping	XOR distributed	Cheaper than shadowing	Moderate	Good

- HSZ40 provides striping capabilities that can be combined with OpenVMS volume shadowing. It provides support for RAID levels 0, 3, and 5.
- Performance trade-off is based on stripe chunk size. A larger chunk size improves throughput for large requests, while a smaller chunk size improves response time for many smaller requests.

Files-11 ODS-2 Concepts

Module 14

Module Outline

Objectives

This module introduces the concepts and structure of an OpenVMS Files-11 file system. Students will be able to describe the ten reserved files created on a Files-11 volume when it is initialized. Students will also understand the steps involved in opening a file on the OpenVMS operating system.

Topics

- Files-11 Terminology
- Directory Concepts
- File Open Operation

Files-11 Terminology

Files-11 ODS-2 is a file structure used to maintain OpenVMS disk volumes.

- Extension of Files-11 ODS-1 is used to maintain the PDP-11 RSX operating system's disk volumes.
- Software supporting Files-11 is called XQP.

Volumes

- Collection of logical blocks describing disk.
- Volumes start at **LBN 0** and grow to **LBN X-1**, where **X** is the size of volume in blocks.

Files

- Collection of virtual blocks allocated from volume.
- All files start at **VBN 1**.

Extents

- Collection of logically contiguous blocks allocated to the same file.

Mapping Pointers

- To access a file, you must be able to locate the description of extents that form file.
- This description is called a *mapping pointer*.
- A mapping pointer is made up of two pieces of information describing the location of extent:
 - Starting logical block number
 - Count of the number of blocks in extent

File Headers

- Contain a description of the file, including owner **UIC**, file name, file protection and so on.
- Contain mapping pointers describing the location of extents that make up the file.
- To open the file, information from the file header must be read into memory.

Clusters

- The *cluster* size of volume defines the number of contiguous blocks allocated to a file, when file is extended by 1 to cluster size blocks.
- Extents and files are sized in multiples of cluster size.

File IDs

- File IDs are assigned uniquely and identify how to locate the file header of a file.
- Of the form (**H,S,R**), where:

H:	Identifies the header number of a file.
S:	Identifies the sequence number (number of times the header block has been reused).
R:	Identifies the relative volume number of the volume containing the header for a file.

Directory Concepts

- A *directory* is a file that catalogs file IDs and corresponding file names.
- A directory is created contiguously and must remain contiguous.
- The file header of directory contains a bit, which marks the file as being a directory.
- A directory is a sequential file, containing variable-length records.
 - Records in a directory are sorted in ascending order by file name.
 - A record contains:
 - ◆ Version limit
 - ◆ Flags
 - ◆ Name of file, including name and type portions of file specification
 - ◆ Version number and file ID—this portion of record is repeated for each version of the file
- A directory allows the use of the file specification of a file to locate file ID of a file, and in turn, to locate the file header of the file.
- There are three general categories of directories: MFD, UFD, and SFD.

Master File Directory (MFD)

- Root directory for the volume.
- Only one **MFD** for a given volume or volume set.
- All directory lookups start at **MFD**.
- Has known file ID of (4,4,0).
- Name is `[000000]000000.DIR;1`.

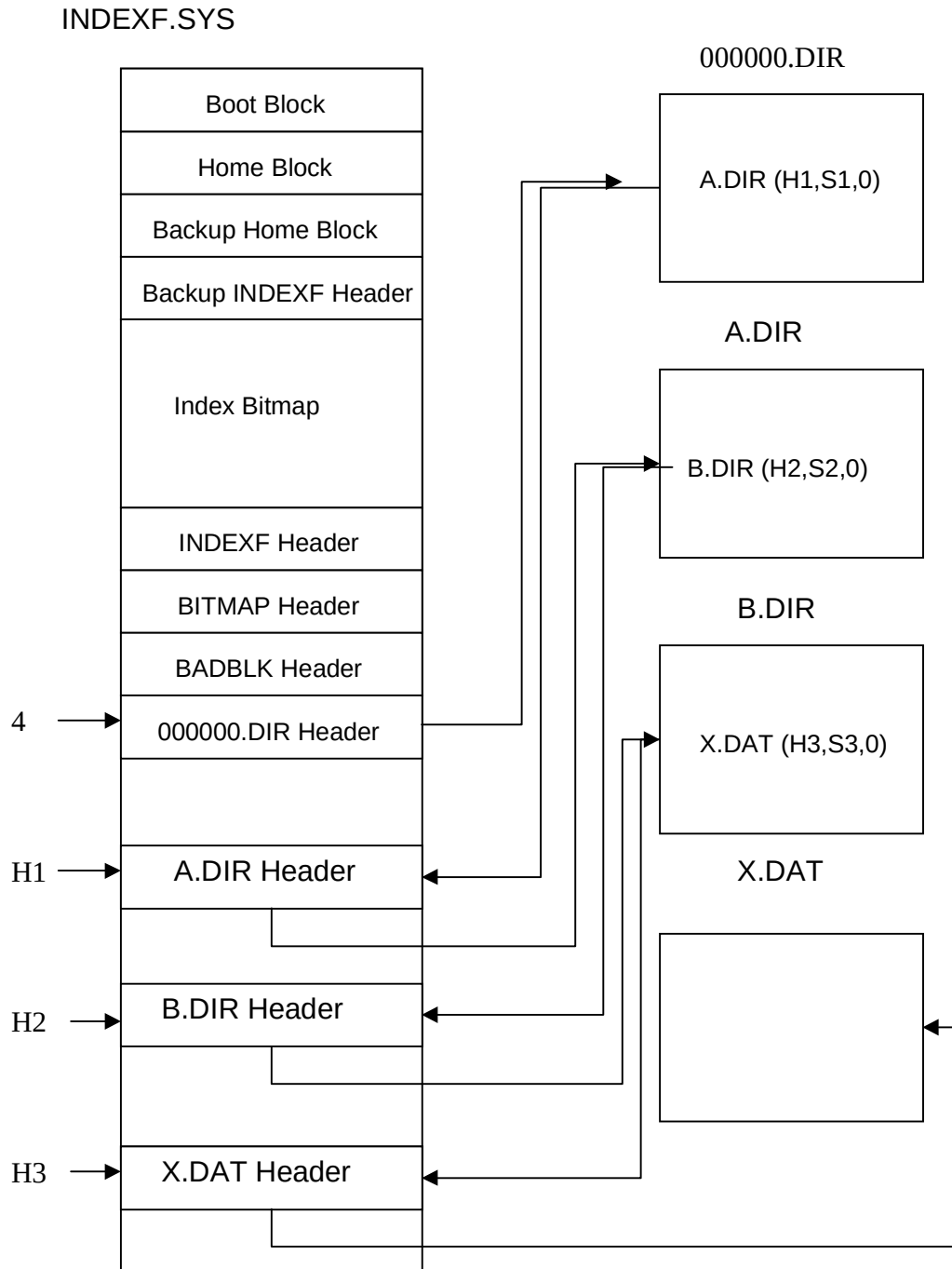
User File Directories (UFD)

- Directories are cataloged in **MFD**.

Sub File Directories (SFD)

- Directories are cataloged in a directory other than **MFD**.

File Open Operation



The following steps detail the steps displayed in the previous diagram and describe the general steps involved in a file open operation:

- The first step involved in a file open is locating MFD, which is the file named **000000.DIR**.
- MFD has a known file id of (4,4,0). Because the first number of the file ID is the header number of the file's header, MFD's header is the fourth header in the index file (**INDEXF.SYS**).
- MFD's header is read into memory.
- Upon locating the header for MFD, its mapping pointer (because directories are contiguous files, only one mapping pointer) is extracted from the header.
- Mapping pointer is used to locate the logical blocks that make up MFD. These blocks are read into memory.
- Because directories have records sorted alphabetically, a modified binary search can be done to locate the record that describes file **A.DIR**.
- Once the record has been located, the file ID for **A.DIR**, which is part of the directory record, is extracted.
- In this case, the file ID is (H1,S1,0), so we must go back to the index file and extract the header block number H1.
- Once the header block for **A.DIR** is read into memory, we can extract the mapping pointer to locate blocks for **A.DIR**.
- **A.DIR** is read and processed in almost the same way as the previous reading of MFD, except that **B.DIR** is located and its file ID is (H2,S2,0).
- Now read the header for **B.DIR** (header block number H2), obtain the mapping pointer, and read blocks for **B.DIR** into memory.
- Search for the record corresponding to **X.DAT** and find that the file ID is (H3,S3,0).
- Read up to the first eight mapping pointers (by default) for **X.DAT** into memory and store them in the data structure called Window Control Block (**WCB**).
- The file is now open and available for processing.

Contiguity and the File System

Module 15

Module Outline

Objectives

This module describes the effects of discontinuous files on disk performance. Methods of analyzing fragmentation through window turns and split transfers will also be described.

Topics

- Window Turns
- Preventing Fragmentation
- Split Transfers
- Window Turns Versus Split Transfers
- Accessing Sequential Files
- Accessing Files Randomly

Contiguity and the File System

Window turns and split transfers are the main indicators of discontinuous files on OpenVMS.

- Although window turns can be minimized without making files contiguous, this is simply masking part of the problem.
- The real problem with discontinuous files is that they incur more overhead due to seeking and rotational latency overhead involved in accessing discontinuous extents that make up the file.

Window Turns

- To locate a file on disk, you must be able to translate **VBNs** in a file to **LBNs** on disk.
- An extent is described by mapping pointers made up of starting **LBN** and count of the number of blocks in extent.
- Mapping pointers were originally stored in the file header of a file.
- When a file is opened, a subset of its mapping pointers is read into memory and stored in a data structure called **WCB**.
- The **WCB** is accessed whenever the driver needs to translate **VBNs** into **LBNs** on I/O transfer.
- Extents described in **WCB** are virtually contiguous in a file.
- The number of mapping pointers that the default **WCB** can hold is eight (**SYSGEN** parameter **ACP_WINDOW+1**).
- If a portion of a file is accessed that is not mapped in **WCB**, **XQP** code is activated to "turn the window" to a portion of the file attempting to access.
- This operation is referred to as a window turn.
- Window turns can be tracked using **\$MONITOR FCP**.
- Costly—require activating **XQP** and can cause I/O to be issued to reread the file header of a file.
- Indicate extra overhead due to seeking and rotational latency being incurred to access blocks not logically adjacent.
- Window turns can be minimized by defining larger **WCB** using:
 - RMS **RTV** field in **FAB** on an individual file open.
 - **/WINDOW** qualifier on mount request for volume to handle a problem on a per-volume basis.
 - Increasing **ACP_WINDOW SYSGEN** parameter to affect a problem on system-wide basis.

- The cost for larger **WCBs** is more nonpaged pool.

Track the additional amount of pool being used by issuing:

```
$SHOW DEVICE/FILES/WINDOW device-name:
```

**Note**

These solutions are not the preferred methods to resolve problems with window turns.

- Real solution is to make discontinuous files contiguous.
- To make a volume that is already fragmented more contiguous, perform image backup and restore, or use one of the various "defragmenters" available in the industry.

Preventing Fragmentation

The best method for preventing fragmentation is to create files using the Contiguous-Best-Try or Contiguous options and preallocating space for the file.

Contiguous-Best-Try

- Tells the file system to scan volume bitmap looking for the three largest extents.
- If request for blocks is not satisfied, first fit allocation is performed.
- The trade-off is more contiguous files rather than a more costly allocation of blocks.

Contiguous

- Tells the file system to get blocks contiguously or return error status.
- Can incur more overhead scanning of the bitmap file for contiguous extent and can hamper functional characteristics of the application.
- Get the same effect once the file has been created by using DCL command **\$COPY/CONTIGUOUS**.

Extent Cache

If neither the Contiguous or Contiguous-Best-Try option is used, blocks are allocated from extent cache (described later) or by using first fit lookup.

- Quicker and may be the preferred method for creating "scratch" files.
- None of these methods are effective if space for the file is not properly preallocated.
- If there is no control over the amount of preallocation of blocks for the file, you can increase default extend size, assuming application has not defined extend size in the opening of file.
- By allocating blocks in larger chunks, you are more likely to grab contiguous extents.
- To increase the number of blocks allocated on extent of file, use:
 - **\$SET RMS/EXTEND=*n*** on a per-process basis.
 - Larger setting for **SYSGEN** parameter **RMS_EXTEND** on system-wide basis.

Split Transfers

- I/O to or from file must be virtually contiguous, but need not be logically contiguous.
- If I/O is not logically contiguous, the transfer is passed to logically contiguous "parcels" to driver.
- The first parcel counts as direct I/O.
 - Subsequent parcels count as split transfers.
 - Split transfers can be traced using **\$MONITOR IO**.
 - The only way to minimize split transfers is to make files more contiguous using the same methods described under window turns.

Window Turns Versus Split Transfers

- Both window turns and split transfers indicate that problems with fragmentation are occurring.
 - Split transfers are more likely to occur when transfers are larger in size.
 - This happens most often when accessing sequential files.
- Window turns are more likely to occur on randomly accessed files - I/Os tend to be smaller.
 - Also more likely to occur in randomly accessed files because of the implied poor locality of random access.
 - Rarely touch records virtually adjacent when accessing files randomly.
 - Window turns commonly beget more window turns when using random access.
- The next example illustrates a high split transfer rate (**8.97/sec**) in **\$MONITOR IO**, with a lower window turn rate (**1.15/sec**) displayed in **\$MONITOR FCP**.
 - Results are due to accessing a fragmented sequential file sequentially.
- The following example shows no split transfers, with a high window turn rate (**71.65/sec**).
 - Due to accessing an extremely fragmented file randomly, with small I/O buffers.

Accessing Sequential Files

\$ MONITOR IO, FCP

VAX/VMS Monitor Utility

I/O SYSTEM STATISTICS

on node BRUDEN

9-NOV-1999 03:05:50

	CUR	AVE	MIN	MAX
Direct I/O Rate	19.93	19.93	19.93	19.93
Buffered I/O Rate	0.33	0.33	0.33	0.33
Mailbox Write Rate	0.00	0.00	0.00	0.00
Split Transfer Rate	8.97	8.97	8.97	8.97
Log Name Translation Rate	1.00	1.00	1.00	1.00
File Open Rate	0.00	0.00	0.00	0.00
Page Fault Rate	7.97	7.97	7.97	7.97
Page Read Rate	5.31	5.31	5.31	5.31
Page Read I/O Rate	0.66	0.66	0.66	0.66
Page Write Rate	0.00	0.00	0.00	0.00
Page Write I/O Rate	0.00	0.00	0.00	0.00
Inswap Rate	0.00	0.00	0.00	0.00
Free List Size	22918.00	22918.00	22918.00	22918.00
Modified List Size	1067.00	1067.00	1067.00	1067.00

VAX/VMS Monitor Utility

FILE PRIMITIVE STATISTICS

on node BRUDEN

9-NOV-1999 03:05:53

	CUR	AVE	MIN	MAX
FCP Call Rate	0.32	0.33	0.32	0.33
Allocation Rate	0.32	0.16	0.00	0.32
Create Rate	0.00	0.00	0.00	0.00
Disk Read Rate	0.00	0.00	0.00	0.00
Disk Write Rate	0.32	0.16	0.00	0.32
Volume Lock Wait Rate	0.00	0.00	0.00	0.00
CPU Tick Rate	0.65	0.33	0.00	0.65
File Sys Page Fault Rate	0.00	0.00	0.00	0.00
Window Turn Rate	1.63	1.15	0.66	1.63
File Lookup Rate	0.00	0.16	0.00	0.33
File Open Rate	0.00	0.00	0.00	0.00
Erase Rate	0.00	0.00	0.00	0.00

Accessing Files Randomly

\$ MONITOR IO, FCP

VAX/VMS Monitor Utility

I/O SYSTEM STATISTICS

on node BRUDEN

9-NOV-1999 03:06:56

	CUR	AVE	MIN	MAX
Direct I/O Rate	73.60	73.60	73.60	73.60
Buffered I/O Rate	0.33	0.33	0.33	0.33
Mailbox Write Rate	0.00	0.00	0.00	0.00
Split Transfer Rate	0.00	0.00	0.00	0.00
Log Name Translation Rate	1.00	1.00	1.00	1.00
File Open Rate	0.00	0.00	0.00	0.00
Page Fault Rate	6.83	6.83	6.83	6.83
Page Read Rate	4.96	4.96	4.96	4.96
Page Read I/O Rate	0.62	0.62	0.62	0.62
Page Write Rate	0.00	0.00	0.00	0.00
Page Write I/O Rate	0.00	0.00	0.00	0.00
Inswap Rate	0.00	0.00	0.00	0.00
Free List Size	22962.00	22962.00	22962.00	22962.00
Modified List Size	1083.00	1083.00	1083.00	1083.00

VAX/VMS Monitor Utility

FILE PRIMITIVE STATISTICS

on node BRUDEN

9-NOV-1999 03:06:59

	CUR	AVE	MIN	MAX
FCP Call Rate	0.32	0.15	0.00	0.32
Allocation Rate	0.32	0.15	0.00	0.32
Create Rate	0.00	0.00	0.00	0.00
Disk Read Rate	0.00	0.00	0.00	0.00
Disk Write Rate	0.32	0.15	0.00	0.32
Volume Lock Wait Rate	0.00	0.00	0.00	0.00
CPU Tick Rate	0.65	0.31	0.00	0.65
File Sys Page Fault Rate	0.00	0.00	0.00	0.00
Window Turn Rate	70.58	71.65	70.58	72.89
File Lookup Rate	0.00	0.00	0.00	0.00
File Open Rate	0.00	0.00	0.00	0.00
Erase Rate	0.00	0.00	0.00	0.00

Module Outline

Objectives

This module describes the caches associated with Files-11 and the XQP. It also describes performance considerations relating to these caches.

Topics

- File System Caches
- \$MONITOR File
- Shared and Private Caches
- Checking for Shrunk Caches

File System Caches

File system caches information relating to on-disk structure to minimize I/O activity.

- File system's I/O can be monitored by using **\$MONITOR FCP** and by examining disk read and write rates.
- If rates are high, the primary cause could be ineffective cache use.

File System Cache Parameters

- **ACP_DINDXCACHE**
 - Provides two general levels of functionality:
 - ◆ Maintaining preformed directory file control blocks (which can be quickly moved into memory).
 - ◆ Maintaining hash table for lookups of directory data cache information.
 - Relevant **\$MONITOR FILE** field: Directory FCB Cache rates
- **ACP_DIRCACHE**
 - Used to store directory data blocks read into memory by the file system.
 - Blocks can be reaccessed to minimize I/O associated with directory lookups.
 - Relevant **\$MONITOR FILE** field: **Directory Data Cache rates**
- **ACP_HDRCACHE**
 - Used to store file header blocks read from the index file.
 - Blocks can be reaccessed on file opens and window turns to minimize reads of file header blocks.
 - Relevant **\$MONITOR FILE** field: **File Header Cache rates**

■ **ACP_FIDCACHE**

- The file system must locate the free file header when the file is created.
- To support this need, we normally have to read the index file bitmap, scan it, and update it to disk.
- When a file is deleted, the file ID is placed in the file ID cache.
- When a new header is needed, the file ID is extracted from this cache in last in, first out (LIFO) fashion.
- This eliminates two I/Os and CPU time required to scan the index bitmap.
- A potential cost for cache is that, should the system crash, the volume will not be properly dismounted.
- This implies that headers will be marked in use when they really are not in use.
- This is resolved when volume rebuild occurs.
- Relevant \$MONITOR FILE field: **File ID Cache rates**

■ **ACP_EXTCACHE**

- When blocks are allocated for a file, the file system normally must read storage bitmap, scan it, and update it to disk.
- When a volume is mounted and the first write operation to file system occurs, a chunk of volume is preallocated.
- This chunk is defined as **ACP_EXTLIMIT** 1000ths of volume.
- By default, parameter is set to 100, so that 10 percent of the volume is preallocated.
- When blocks are needed for a file, the file system attempts to grab extents described in extent cache to support the request.
- When files are deleted, their extents are added to cache to replenish it.
- Using this cache eliminates any I/Os required to read and update the storage bitmap file and eliminates any CPU time required to scan it.
- A potential cost of this cache is that, should the system crash, the blocks will not be returned to the volume.
- Relevant \$MONITOR FILE field: **Extent Cache rates**

■ **ACP_QUOCACHE**

- Stores disk quota entries for any disks with quotas enabled.
- Using this cache minimizes the need to perform I/Os to disk quota file.
- The potential cost is based on invalid quota entries due to system crash.
- The problem is also resolved through volume rebuild.
- Relevant **\$MONITOR FILE** field: **Quota Cache rates**

■ **ACP_MAPCACHE**

- Stores blocks from the volume bitmap read to support the allocation of blocks.
- Using this cache minimizes the need to read the bitmap file.
- Relevant **\$MONITOR FILE** field: **Map Cache rates**

Examining File System Caching Parameters

```

$
$ mcr sysgen
SYSGEN>  SHOW /ACP

Parameters in use: Active
Parameter Name      Current    Default    Min.      Max.      Unit Dynamic
-----
ACP_MULTIPLE        0          0          0          1 Boolean  D
ACP_SHARE            1          1          0          1 Boolean
ACP_MAPCACHE         8          8          1         -1 Pages   D
ACP_HDRCACHE        54         32          3         -1 Pages   D
ACP_DIRCACHE        54         20          2         -1 Pages   D
ACP_DINDXCACHE      13         25          2         -1 Pages   D
ACP_WORKSET          0          0          0         -1 Pages   D
ACP_FIDCACHE        64         64          0         -1 File-Ids D
ACP_EXTCACHE        64         64          0         -1 Extents  D
ACP_EXTLIMIT       100        100          0        1000 Percent/10 D
ACP_QUOCACHE        30         64          0        2337 Users   D
ACP_SYSACC           9          8          0         -1 Directorie D
ACP_MAXREAD         32         32          1          64 Blocks   D
ACP_WINDOW           7          7          1         -1 Pointers  D
ACP_WRITEBACK        1          1          0          1 Boolean  D
ACP_DATACHECK        2          2          0          3 Bit-mask  D
ACP_BASEPRIO         8          8          4          31 Priority  D
ACP_SWAPFLGS        14         15          0          15 Bit-mask  D
ACP_XQP_RES          1          1          0          1 Boolean
ACP_REBLDSYSD        1          1          0          1 Boolean

SYSGEN>  EXIT
$

```

- This display illustrates examining these parameter values by using the **SYSGEN** utility.
- Caches can be tracked using: **\$MONITOR FILE_SYSTEM_CACHES**

\$MONITOR File

\$ MONITOR FILE_SYSTEM_CACHES

VAX/VMS Monitor Utility
FILE SYSTEM CACHING STATISTICS
on node BRUDEN
9-NOV-1999 03:15:35

		CUR	AVE	MIN	MAX
Dir FCB	(Hit %)	67.00	67.56	67.00	67.00
	(Attempt Rate)	12.09	12.09	12.09	12.09
Dir Data	(Hit %)	62.00	62.16	62.00	62.00
	(Attempt Rate)	12.09	12.09	12.09	12.09
File Hdr	(Hit %)	52.00	52.00	52.00	52.00
	(Attempt Rate)	8.16	8.16	8.16	8.16
File ID	(Hit %)	0.00	0.00	0.00	0.00
	(Attempt Rate)	0.00	0.00	0.00	0.00
Extent	(Hit %)	0.00	0.00	0.00	0.00
	(Attempt Rate)	0.00	0.00	0.00	0.00
Quota	(Hit %)	0.00	0.00	0.00	0.00
	(Attempt Rate)	0.00	0.00	0.00	0.00
Bitmap	(Hit %)	0.00	0.00	0.00	0.00
	(Attempt Rate)	0.00	0.00	0.00	0.00

Interpreting the \$MONITOR File

Parameters relating to given fields have been mentioned in previous discussion.

- When numbers are interpreted, remember that a hit indicates that information was located in cache and I/O was not required.
- If not a hit, I/O was generated.
- Taking 1.00, minus hit percentage (as decimal value), times attempt rate yields the number of I/Os caused by being unable to find information in cache.
- In general, shoot for an 80 - 90+ percent hit rate.
- However, when a cache (such as a bitmap cache) has a low attempt rate (usually because blocks are found in the extent cache), the hit percentage rate is generally unimportant because of the low I/O rate generated through low attempts.

Shared and Private Caches

- The caches: **ACP_EXTCACHE**, **ACP_FIDCACHE**, and **ACP_QUOCACHE** are private to volume and can be increased on per-volume basis. They are allocated from nonpaged pool.
- The caches: **ACP_DINDXCACHE**, **ACP_DIRCACHE**, **ACP_HDRCACHE**, and **ACP_MAPCACHE** are shared by all volumes and are allocated from paged pool.
- These caches become private if volume is mounted using the **/PROCESSOR=UNIQUE** qualifier.
- To support these caches, you must have sufficient contiguous paged pool.
- It is important to track the parameter **PAGEDYN** when these cache sizes are increased.
- Be cautious—cache sizes are in units of pages, while **PAGEDYN** is sized in units of bytes.
- If there is insufficient paged pool, caches will shrink substantially.
- Suspect that caches are shrunken if shared caches have a low hit percentage rate, while private caches have a high hit percentage rate.
- To confirm suspicions, sum the size of shared caches.
- If the sum matches the value in field "**Maximum buffers in FCP cache**" in the **\$SHOW DEVICE/FULL** display on disk, caches have been sized properly.
- If values do not match, the value in field "**Maximum buffers in FCP cache**" will read 14, yielding extremely poor cache hit rates for shared caches.
- The problem can be corrected by either increasing the size of **PAGEDYN** or decreasing shared cache sizes.

Checking for Shrunk Caches

The following display illustrates steps involved in checking for shrunk caches.

- Note that the sum of caches **ACP_DINDXCACHE**, **ACP_DIRCACHE**, **ACP_HDRCACHE**, and **ACP_MAPCACHE** total 129, which is the same value shown in the maximum buffers in **FCP** cache field from the sample **\$SHOW DEVICE/FULL** command on disk. The caches are not shrunk.

```
$
$ type cache.com
$x = f$getsyi("ACP_DINDXCACHE")+F$GETSYI("ACP_DIRCACHE")
$x = F$GETSYI("ACP_HDRCACHE") + F$GETSYI("ACP_MAPCACHE") + X
$write sys$output "The sum of the shared cache sizes is ",x
$ @cache
The sum of the shared cache sizes is 129
$ show device/full sys$sysdevice:
```

Disk \$1\$DKA300: (BRUDEN), device type Generic SCSI disk, is online, mounted,
file-oriented device, shareable, error logging is enabled.

Error count	0	Operations completed	9560
Owner process	" "	Owner UIC	[SYSTEM]
Owner process ID	00000000	Dev Prot	S:RWED,O:RWED,G:RWED,W:RWED
Reference count	127	Default buffer size	512
Total blocks	1015812	Sectors per track	63
Total cylinders	1466	Tracks per cylinder	11
Allocation class	1		
Volume label	"BRUDEN"	Relative volume number	0
Cluster size	3	Transaction count	186
Free blocks	563649	Maximum files allowed	126976
Extend quantity	5	Mount count	1
Mount status	System	Cache name	"_ \$1\$DKA300:XQPCACHE"
Extent cache size	64	Maximum blocks in extent cache	56364
File ID cache size	64	Blocks currently in extent cache	2856
Quota cache size	0	Maximum buffers in FCP cache	129

Volume status: subject to mount verification, file high-water marking,
write-
through caching enabled.

```
$
```

RMS Structures and Design Considerations

Module 17

Module Outline

Objectives

This module describes RMS file organizations and their impact on performance. Particular emphasis will be placed on the internal design of indexed files. Students will understand the impact of bucket splits and internal fragmentation of indexed files on performance.

Topics

- RMS File Concepts
- Summary of RMS File Organizations
- Sequential and Relative File Structures
- Indexed File Organization for a File with No Alternate Keys
- RMS Interfaces for Affecting File Structure
- Bucket Splits
- Indexed File Growth
- Factors Affecting Bucket Splits
- Duplicate Keys for Data Records
- Compression

RMS File Concepts

The Record Management Services apply a record structure to files. RMS record processing allows a programmer to read, write, update, and delete small, but meaningful, pieces (*records*) of files.

RMS supports:

- File organizations (methods to coordinate the access of records within files).
 - Sequential—for accessing one record at a time, first to last.
 - Relative—for accessing randomly by record number.
 - Indexed—for accessing randomly by key (some smaller piece of a record).
- Record organizations (methods for describing the data within the record).
 - Variable length—data can vary in size on a record-by-record basis (2-byte prefix at the beginning of the record).
 - Fixed length—record address is computed based on fixed size.
 - Variable with fixed control—variable length record with fixed size prefix, which can be ignored or included in read.
 - Stream—record is made up of a *stream* of bytes, terminated by a single or sequence of characters, usually carriage return or linefeed.
- Record Access Methods (methods to obtain records from files).
 - Sequential—processing records one at a time from first to last.
 - Random by record number—as the name implies, a record number is used to compute the location of the record.
 - Random by key—record is located through a binary tree type lookup using a portion of a record to key the lookup.
 - Random by record file address (RFA)—a 6-byte entity, the RFA identifies the precise location of the record in the file.

General RMS terms include:

- Buffer—an area of memory used to store blocks read from or to be written to a file.
- Bucket—unit of allocation for the storage of records in indexed and relative files. Also used to determine I/O buffer sizes.
- Multiblock count—size of an I/O buffer for a sequential file.
- Record cell—fixed size *container* for records in a Relative file.

Summary of RMS File Organizations

Access Mode	Sequential	Relative	Indexed
Sequential	Yes	Yes	Yes
Random by record number	No, unless fixed length records	Yes	No
Random by key	No	No	Yes
Random by record file address (disk only) (RFA)	Yes	Yes	Yes

Record Format	Sequential	Relative	Indexed
Fixed length	Yes	Yes	Yes
Variable length	Yes	Yes	Yes
VFC	Yes	Yes	No
Stream	Yes	No	No
Undefined	Yes	No	No

Record Format	Sequential	Relative	Indexed
I/O unit	Multiblock count (1 or more blocks)	Bucket	Bucket

All organizations support all RMS operations, except:

- Only sequential files support the truncate operation.
- Sequential files do not support delete operations.

File Structures

Sequential File Organization

- Sequential files have no RMS-related overhead beyond the record format.
- Space is allocated for sequential files in blocks (multiples of cluster factor of volume).
- Read in quantities of blocks is defined by file, process, or system multiblock count.

Relative Files

- Space is allocated (beyond prologue) and I/O is performed in units of buckets.
- Bucket size is defined in the file header.
- Cell format:

PB	Record 1		PB	Record 2		...
----	----------	--	----	----------	--	-----

“PB” is the “presence byte”, indicating whether the cell has ever been written and whether the record has been deleted.

Indexed Sequential Files

- Indexed files support sequential or random records by key access.
- Must define primary key.
- Primary keys can be described in ascending or descending order.
- Primary key can be segmented into a maximum of eight different portions of the record.
- Indexed files can have up to 255 alternate keys.
- RMS supports three "*prologues*" defining usage and types of keys. The most current and common is prologue 3.

Prologue 3 supports all available key types, as well as data, key, and index compression and reclamation of empty buckets without creating a new data file (**\$CONVERT/RECLAIM**).

- *Areas* allow portions of indexed files to be treated as separate entities with separate bucket sizes, allocation sizes, extend quantities, and contiguity options.

RMS Interfaces for Affecting File Structure

General File-Oriented Routines

SYSS\$CREATE	Create (or open) file.
SYSS\$OPEN	Open file for processing.
SYSS\$CLOSE	Close file.

Record Processing Routines

SYSS\$CONNECT	Define attempted access to record stream to RMS , and make stream available for record processing.
SYSS\$DISCONNECT	Disable given record stream.
SYSS\$GET	Read record from file associated with record stream.
SYSS\$PUT	Write record to file associated with record stream.

Structures Defining File/Record Characteristics to Routines

File Access Block (FAB)	Defines file characteristics for creating or accessing file.
Record Access Block (RAB)	Defines characteristics of record stream used for record processing.
Name Block (NAM)	Defines characteristics for advanced file name processing.
Extended Attributes	Defines extended file or record attributes such as space allocation, date and time stamps in file header, header characteristics of indexed files, and terminal processing options.
Block (XAB)	XAB may be associated with FAB or RAB , depending on usage.

Bucket Splits

- A bucket split occurs when RMS attempts to insert a record into a bucket that is already full.
- Because there is no place to store the record in the existing bucket, a new bucket is allocated and a subset of the records (usually half) is removed from the initial bucket into the new bucket.

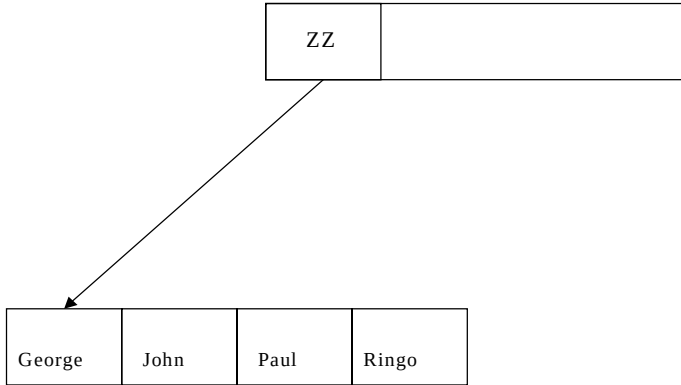
Record Reference Vectors (RRVs)

- The record file address (**RFA**) of a record must be preserved throughout the life of the record.
- Preservation is necessary for user applications that may be using **RFA** to access records and for the secondary index data record used to locate data buckets when record is accessed by alternate key.
- The Secondary Index Data Record (**SIDR**) is not updated to point to "*split*" location of data record, because this would incur substantial overhead and the **RFAs** must be preserved for user applications using **RFA** access.
- **RRV** contains the **RFA** of the split location of the data record.

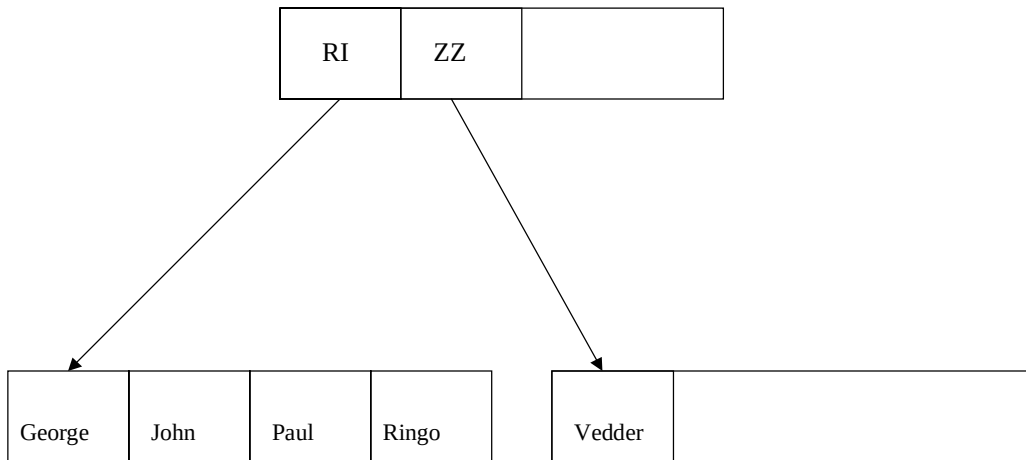
Index Bucket Splits

- When a data bucket is split, a new key record must be added to locate the data bucket.
- As more index records are added, the index bucket may be forced to split.
- This can force new index levels to be incorporated in the index file structure.

Indexed File Growth

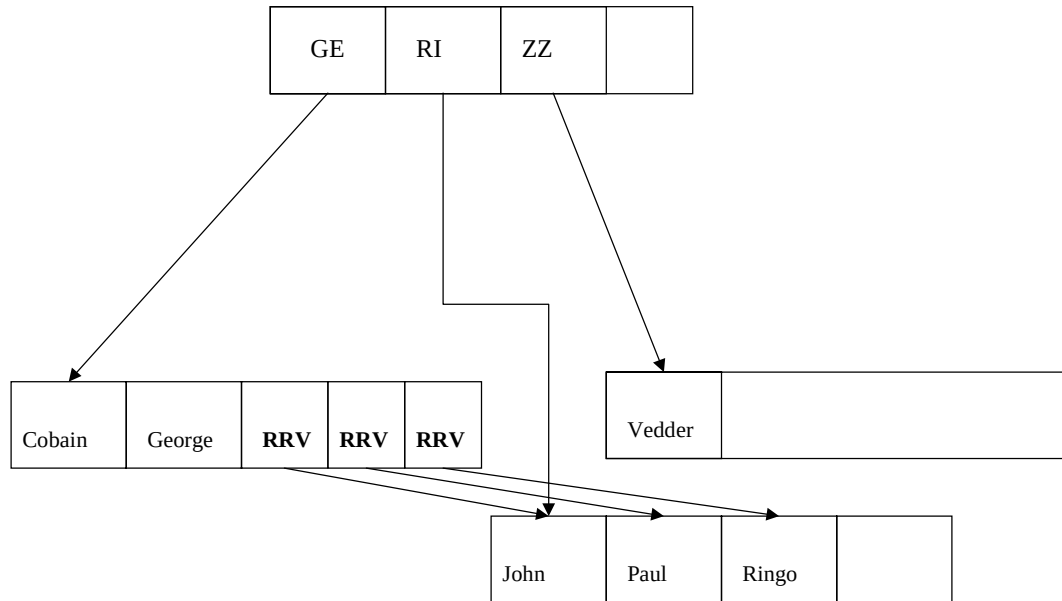


After inserting VEDDER



This shows the bucket split after the addition of the record COBAIN.

After inserting COBAIN



Factors Affecting Bucket Splits

Population Techniques

- Indexed files are usually maintained in ascending key order.
- Populate the indexed file in ascending order of the primary key.
- Specify descending key data types, if necessary.

Fill Factor

- Defines the percentage of bucket to fill prior to moving to the next bucket on initial load of the file.
- When file processing includes many additions spread normally over keys, you may want to use a smaller fill factor to reserve space for record additions in the bucket.
- Cost for a small fill factor is temporary or long-term wasted space within a bucket.
- You can maintain separate fill factors for data and index buckets.

Bucket Size

- The larger the bucket, the more records it can hold.
- Potential cost:
 - Disk transfer time (potentially made up for in seek time).
 - Data bucket search time (CPU dependent).
 - Memory for buffer (buffers sized to hold largest bucket).
 - Bucket lock contention.
 - Potential data loss using deferred write.

Number of Record Additions Over Time

- A stable file (one with no additions) will have no bucket splits other than those potentially caused due to initial load.
- A file with many additions over time may have to be converted (**\$CONVERT**) regularly.

Duplicate Keys for Data Records

Data records with duplicate keys have only one index record pointing to them.

- If there are more records than fit into one bucket, an overflow bucket is allocated.
 - The next bucket pointer in the data bucket header permits scanning from one data record to the next.
 - Data records scanned sequentially.
- On bucket split, RMS attempts to keep duplicate key records in the same bucket, making the bucket split more involved than one without duplicate keys.

Compression

RMS supports compression of keys and data items to conserve space in buckets.

- Trades off CPU time to "*decompress*" records rather than add more data/keys per bucket.
- More data/keys per bucket minimizes number of levels and buckets while maximizing I/O (more data read in per I/O).

Data Key Compression

- Criteria: String type and string at least 6 bytes long.
- Implementation: Leading characters common with previous are suppressed. Trailing duplicates, such as spaces, are suppressed.

Data Record Compression

- Criteria: five or more contiguous bytes with repeating characters.
- Implementation: characters preceded with 2-byte count of uncompressed characters, followed by uncompressed characters, compression character, and compression count. This is repeated.

Index Compression

- Identical to key compression, except in the index bucket.
- Negative performance implications:
 - If index compression is enabled, disable the binary search of the index bucket.
 - Compression can be negative; can cost more in overhead than the space gained in compression.

Module Outline

Objectives

This module describes the use of file definition language (FDL) files in producing optimal file formats. The CONVERT utility will be described in conjunction with the creation and conversion of indexed files for more optimal performance.

Topics

- File Definition Language
- Techniques for Creating Data Files Using an FDL File
- Creating a Data File from an FDL File
- Optimizing RMS Indexed Files
- Examining File Structures

File Definition Language

File Definition Language (FDL) defines a language to produce an ASCII text file describing RMS attributes of a file to RMS utilities and utility routines.

- Divided into sections (or primary attributes).
- Sections are broken into secondary attributes describing specific RMS attributes to be used in file creation or possibly during run-time processing.

Creating an FDL File

- The following command creates an FDL file based on characteristics of the input data file:

```
$ANALYZE/RMS/FDL data-file-specification
```

- FDL file has the same file specification as data file except that it has **.FDL** file type.

Creating a Text File

- Create with FDL attributes desired using a text editor of choice, assuming proper FDL rules.
- Or, use **ADD**, **DELETE**, **MODIFY**, **VIEW** and **EXIT** commands in **\$EDIT/FDL**.

Techniques for Creating Data Files Using an FDL File

- The **\$EDIT/FDL** command:
 - **\$EDIT/FDL file.fdl**
 - **\$EDIT/EDIT/SCRIPT = script-title**
- **\$EDIT/FDL** command will provide a menu of items with **INVOKE** menu item providing options corresponding to script-titles used with **/SCRIPT** qualifier. Menu items include:

Menu Item	Description
ADD_KEY	Allows adding key definition for the file produced from the FDL file
DELETE_KEY	Allows deleting highest key definition.
INDEXED	Uses prompting to obtain information about section definitions for indexed file creation.
OPTIMIZE	Allows producing an optimal FDL file for file creation of data file based on the input FDL file produced from \$ANALYZE/RMS/FDL .
RELATIVE	Uses prompting to obtain information for creation of FDL file for relative file creation.
SEQUENTIAL	Prompting for producing FDL for creation of a sequential file.
TOUCHUP	Prompting for modification of key characteristics.

Sample EDIT/FDL Session

```
$
$ edit/fdl/script=indexed ind_test.fdl

      Parsing Definition File

_DKA300:[ELLIS]IND_TEST.FDL; will be created.

      Indexed Script

Target disk volume Cluster Size      (1-2Giga)[3]  :
Number of Keys to Define      (1-255)[1]      :
(Line Fill Key Record Init Add)
Graph type to display          (Keyword)[Line]    :

Number of Records that will be Initially Loaded
into the File      (0-2Giga)[-] : 300000
(Fast_Convert NoFast_Convert RMS_Puts)
Initial File Load Method      (Keyword)[Fast]    :
Number of Additional Records to be Added After
the Initial File Load      (0-2147183645)[0]: 40000
Will Additional Records Typically be Added in
Order by Ascending Primary Key      (Yes/No)[No]:
Will Added Records be Distributed Evenly over the
Initial Range of Pri Key Values      (Yes/No)[No] : yes
Key 0 Load Fill Percent      (50-100)[100] :
(Fixed Variable)
Record Format      (Keyword)[Var] :
Mean Record Size      (1-32224)[-]: 300
Maximum Record Size      (0,300-32224)[0] : 400
(Bin2 Bin4 Bin8 Int2 Int4 Int8 Decimal String Collated
 Dbin2 Dbin4 Dbin8 Dint2 Dint4 Dint8 Ddecimal Dstring Dcollated)
Key 0 Data Type      (Keyword)[Str] :
Key 0 Segmentation desired      (Yes/No)[No] :
Key 0 Length      (1-255)[-] : 15
Key 0 Position      (0-385)[0] :
Key 0 Duplicates allowed      (Yes/No)[No] :
File Prolog Version      (0-3)[3] :
Data Key Compression desired      (Yes/No)[Yes] :
Data Record Compression desired      (Yes/No)[Yes] :
Index Compression desired      (Yes/No)[Yes] :
```

```

      *|
      9|
      8|
Index   7|
      6|
Depth   5|
      4|  4
      3|   3 3 3 3 3
      2|           2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2
      1|
      +-+ - - - - + - - - - + - - - - + - - - - + - - - - + - - - - + - - - - + - - - - +
        1       5       10      15      20      25      30      32
        Bucket Size (number of blocks)

```

```

PV-Prolog Version      3 KT-Key  0 Type      String EM-Emphasis Flatter ( 9)
DK-Dup Key  0 Values   No KL-Key  0 Length      15 KP-Key  0 Position      0
RC-Data Record Comp    0% KC-Data Key Comp      0% IC-Index Record Comp    0%
BF-Bucket Fill          100% RF-Record Format Variable RS-Mean Record Size   300
LM-Load Method Fast_Conv IL-Initial Load   300000 AR-Added Records   40000
(Type "FD" to Finish Design)
Which File Parameter (Mnemonic)[refresh] : fd

```

```

Text for FDL Title Section (1-126 chars)[null] :
Data File file-spec (1-126 chars)[null] : ind_test.dat

```

```

(Carriage_Return FORTRAN None)
Carriage Control (Keyword)[Carr] :
Emphasis Used In Defining Default: ( Flatter_files )
Suggested Bucket Sizes: ( 3 9 18 )
Number of Levels in Index: ( 3 2 2 )
Number of Buckets in Index: ( 1222 119 31 )
Pages Required to Cache Index: ( 3666 1071 558 )
Processing Used to Search Index: ( 24 18 20 )
Key 0 Bucket Size (1-63)[9] :
Key 0 Name (1-32 chars)[null] :
Global Buffers desired (Yes/No)[No] :
The Depth of Key 0 is Estimated to be No Greater
than 2 Index levels, which is 3 Total levels.
Press RETURN to continue (^Z for Main Menu)

```

```

(Add Delete Exit Help Invoke Modify Quit Set View)
Main Editor Function (Keyword)[Help] : exit

```

```

_DKA300:[ELLIS]IND_TEST.FDL;1 41 lines
$

```

```
$ type ind_test.fdl
IDENT          "21-MAR-1999 02:37:39  OpenVMS FDL Editor"

SYSTEM
  SOURCE              "OpenVMS"

FILE
  NAME                "ind_test.dat"
  ORGANIZATION        indexed

RECORD
  CARRIAGE_CONTROL    carriage_return
  FORMAT              variable
  SIZE                400

AREA 0
  ALLOCATION            232878
  BEST_TRY_CONTIGUOUS yes
  BUCKET_SIZE         9
  EXTENSION            58221

AREA 1
  ALLOCATION            1080
  BEST_TRY_CONTIGUOUS yes
  BUCKET_SIZE         9
  EXTENSION            270

KEY 0
  CHANGES             no
  DATA_AREA           0
  DATA_FILL           100
  DATA_KEY_COMPRESSION yes
  DATA_RECORD_COMPRESSION      yes
  DUPLICATES           no
  INDEX_AREA           1
  INDEX_COMPRESSION    no
  INDEX_FILL           100
  LEVEL1_INDEX_AREA    1
  PROLOG               3
  SEG0_LENGTH          15
  SEG0_POSITION        0
  TYPE                 string

$
```

Creating a Data File from an FDL File

Once an FDL file is created, you can use several techniques to create a data file, depending on whether this is an initial file creation or optimization or whether the file is desired at run time.

- The following command produces a "shell" file with no contents, but using FDL characteristics specified in FDL text file:
`CREATE/FDL=file.fdl data-file`
- Populate the file from the application.
- The following command uses the FDL specification to define characteristics for the output data file:
`$CONVERT/FDL=fdl-file input-datafile(s) output-datafile`

Optimizing RMS Indexed Files

Over time, additions and deletions performed on an indexed file can degrade performance characteristics of the file, primarily due to bucket splitting and the potential introduction of many levels.

General Optimization Techniques

- The following command has no major impact on performance characteristics of the file:
`$CONVERT/RECLAIM file`
- Reclaim option reclaims "*empty*" buckets in which all records have been deleted.
- Allows for reuse of these buckets and possibly minimizes growth.
- The following command tends to improve performance characteristics of the file—records are reinserted in order of primary key:
`$CONVERT old new`
- Eliminates all bucket splits, with the exception of those occurring on secondary index data records for alternate keys.

FDL Optimization

- Steps:
`$ANALYZE/RMS/FDL OLD.DAT`

`$EDIT/FDL/ANALYSIS=OLD.FDL NEW.FDL`
`use OPTIMIZE script`

`$CONVERT/NOSORT/FDL=NEW.FDL OLD.DAT NEW.DAT`
- Allocates enough space to hold all index buckets.
- Extends size set to roughly 25 percent of initial allocation.
- Calculates default bucket size by calculating index depth for various bucket sizes and determining the most common index depth.
 - Choosing smaller buffers, it chooses the smallest bucket size to create the most common index depth.
- Global buffers are sized to cache as much of the primary key index as is reasonable.
- Will print an error message if there are more than 32 levels.

**Note**

The same bucket split optimizations occur on optimization steps as in straight convert.

Examining File Structures

- To generate statistical information on the characteristics of a file:
\$ANALYZE/RMS/STATISTICS
- This will display information on the file including:
 - Number of blocks allocated per area
 - Number of index levels for each key
 - Count of data records
 - Mean data record length
 - Compression percentages

Example Using RMS Utilities to Tune an Indexed File Structure

Notes on the test:

- In the first test, a file was populated randomly with approximately 100,000 records.
- In the second test, the file was converted with no special optimizations.
- In the third test, the file was created through a convert with an optimized FDL file.

```
$ sh status
Status on 14-MAR-1999 02:24:30.84 Elapsed CPU : 0 00:00:00.32
Buff. I/O : 85 Cur. ws. : 2000 Open files : 0
Dir. I/O : 16 Phys. Mem. : 1200 Page Faults : 146
$ run read_seq_idx
$ sh stat
Status on 14-MAR-1999 02:29:17.24 Elapsed CPU : 0 00:01:03.69
Buff. I/O : 102 Cur. ws. : 2000 Open files : 0
Dir. I/O : 17184 Phys. Mem. : 1056 Page Faults : 195
$
$ ana/rms/fdl indexed.dat
$
$
$ type indexed.fdl
IDENT "14-MAR-1999 02:29:59 OpenVMS ANALYZE/RMS_FILE Utility"

SYSTEM
SOURCE OpenVMS

FILE
ALLOCATION 59748
BEST_TRY_CONTIGUOUS no
BUCKET_SIZE 2
CLUSTER_SIZE 3
CONTIGUOUS no
EXTENSION 0
FILE_MONITORING no
GLOBAL_BUFFER_COUNT 0
NAME "_DKA300:[ELLIS.RMS]INDEXED.DAT;1"
ORGANIZATION indexed
OWNER [ELLIS]
PROTECTION (system:RWED, owner:RWED, group:RE, world:)

RECORD
BLOCK_SPAN yes
CARRIAGE_CONTROL carriage_return
FORMAT variable
SIZE 220
```

```
AREA 0
  ALLOCATION          59748
  BUCKET_SIZE        2
  EXTENSION           0

KEY 0
  CHANGES            no
  DATA_KEY_COMPRESSION yes
  DATA_RECORD_COMPRESSION yes
  DATA_AREA          0
  DATA_FILL          100
  DUPLICATES          no
  INDEX_AREA          0
  INDEX_COMPRESSION   yes
  INDEX_FILL          100
  LEVEL1_INDEX_AREA   0
  NAME                ""
  NULL_KEY            no
  PROLOG              3
  SEG0_LENGTH         12
  SEG0_POSITION       0
  TYPE                string

KEY 1
  CHANGES            no
  DATA_KEY_COMPRESSION yes
  DATA_AREA          0
  DATA_FILL          100
  DUPLICATES          yes
  INDEX_AREA          0
  INDEX_COMPRESSION   yes
  INDEX_FILL          100
  LEVEL1_INDEX_AREA   0
  NAME                ""
  NULL_KEY            no
  SEG0_LENGTH         20
  SEG0_POSITION       12
  TYPE                string

ANALYSIS_OF_AREA 0
  RECLAIMED_SPACE     0
```

ANALYSIS_OF_KEY 0

DATA_FILL	68
DATA_KEY_COMPRESSION	23
DATA_RECORD_COMPRESSION	14
DATA_RECORD_COUNT	99188
DATA_SPACE_OCCUPIED	34004
DEPTH	3
INDEX_COMPRESSION	18
INDEX_FILL	74
INDEX_SPACE_OCCUPIED	528
LEVEL1_RECORD_COUNT	17002
MEAN_DATA_LENGTH	120
MEAN_INDEX_LENGTH	14

ANALYSIS_OF_KEY 1

DATA_FILL	69
DATA_KEY_COMPRESSION	43
DATA_RECORD_COUNT	98595
DATA_SPACE_OCCUPIED	5164
DEPTH	3
DUPLICATES_PER_SIDR	0
INDEX_COMPRESSION	26
INDEX_FILL	64
INDEX_SPACE_OCCUPIED	134
LEVEL1_RECORD_COUNT	2582
MEAN_DATA_LENGTH	18
MEAN_INDEX_LENGTH	22

\$

\$ **rename indexed.dat indexed.old1**

\$ **convert indexed.old1 indexed.dat**

\$ **sh status**

Status on	14-MAR-1999 02:44:50.15	Elapsed CPU :	0 00:03:14.65
Buff. I/O :	376	Cur. ws. :	2000
Open files :	0		
Dir. I/O :	81194	Phys. Mem. :	944
Page Faults :	887		

\$ **run read_seq_idx**

\$ **sh stat**

Status on	14-MAR-1999 02:46:12.35	Elapsed CPU :	0 00:04:02.62
Buff. I/O :	421	Cur. ws. :	2000
Open files :	0		
Dir. I/O :	93296	Phys. Mem. :	880
Page Faults :	938		

\$

Notes on Optimizations

- The randomly populated file accessed sequentially had the following performance metrics:
 - Elapsed time: 4:46.40
 - Direct I/Os: 17,168
- The file created through simple conversion had the following metrics:
 - Elapsed time: 1:22.20
 - Direct I/Os: 12,102
- Items of note in analysis:
 - Bucket fill on randomly populated file:
 - ♦ Data fill: 68%
 - ♦ Key 0 fill: 74%
 - Caused by bucket splits.
 - What is not seen:
 - ♦ Internal fragmentation due to bucket splits
 - ♦ External fragmentation due to extending the file

Example Using RMS Utilities to Tune an Indexed File Structure (continued)

```
$ ana/rms/fdl indexed.dat
$ type indexed.fdl
IDENT  "14-MAR-1999 02:46:44  OpenVMS ANALYZE/RMS_FILE Utility"

SYSTEM
      SOURCE                      OpenVMS

FILE
      ALLOCATION                    59748
      BEST_TRY_CONTIGUOUS         no
      BUCKET_SIZE                  2
      CLUSTER_SIZE                 3
      CONTIGUOUS                   no
      EXTENSION                     0
      FILE_MONITORING              no
      GLOBAL_BUFFER_COUNT           0
      NAME                         "_DKA300:[ELLIS.RMS]INDEXED.DAT;1"
      ORGANIZATION                 indexed
      OWNER                        [ELLIS]
      PROTECTION                   (system:RWED, owner:RWED, group:RE, world:)

RECORD
      BLOCK_SPAN                   yes
      CARRIAGE_CONTROL             carriage_return
      FORMAT                       variable
      SIZE                         220

AREA 0
      ALLOCATION                    59748
      BUCKET_SIZE                  2
      EXTENSION                     0
```

KEY 0

CHANGES	no
DATA_KEY_COMPRESSION	yes
DATA_RECORD_COMPRESSION	yes
DATA_AREA	0
DATA_FILL	100
DUPLICATES	no
INDEX_AREA	0
INDEX_COMPRESSION	yes
INDEX_FILL	100
LEVEL1_INDEX_AREA	0
NAME	""
NULL_KEY	no
PROLOG	3
SEG0_LENGTH	12
SEG0_POSITION	0
TYPE	string

KEY 1

CHANGES	no
DATA_KEY_COMPRESSION	yes
DATA_AREA	0
DATA_FILL	100
DUPLICATES	yes
INDEX_AREA	0
INDEX_COMPRESSION	yes
INDEX_FILL	100
LEVEL1_INDEX_AREA	0
NAME	""
NULL_KEY	no

SEG0_LENGTH	20
SEG0_POSITION	12
TYPE	string
ANALYSIS_OF_AREA 0	
RECLAIMED_SPACE	0
ANALYSIS_OF_KEY 0	
DATA_FILL	93
DATA_KEY_COMPRESSION	25
DATA_RECORD_COMPRESSION	14
DATA_RECORD_COUNT	99188
DATA_SPACE_OCCUPIED	23834
DEPTH	3
INDEX_COMPRESSION	16
INDEX_FILL	97
INDEX_SPACE_OCCUPIED	286
LEVEL1_RECORD_COUNT	11917
MEAN_DATA_LENGTH	120
MEAN_INDEX_LENGTH	14
ANALYSIS_OF_KEY 1	
DATA_FILL	97
DATA_KEY_COMPRESSION	43
DATA_RECORD_COUNT	98595
DATA_SPACE_OCCUPIED	4052
DEPTH	2
DUPLICATES_PER_SIDR	0
INDEX_COMPRESSION	26
INDEX_FILL	97
INDEX_SPACE_OCCUPIED	68
LEVEL1_RECORD_COUNT	2026
MEAN_DATA_LENGTH	20
MEAN_INDEX_LENGTH	22
\$	


```
$ editx/fdl/script=optimize indexed.fdl
```

```
    Parsing Definition File
```

```
    Definition Parse Complete
```

```
    Optimize Script
```

An Input Analysis File is necessary for Optimizing Keys.

```
Analysis File file-spec      (1-126 chars)[null]: indexed.fdl
```

```
Parsing Analysis File
```

```
Analysis Parse Complete
```

The Definition of Key 0 will be replaced.

Press RETURN to continue (^Z for Main Menu)

```
(Line Fill Key Record Init Add)
```

```
Graph type to display      (Keyword)[Line]      :
```

```
Number of Records that will be Reloaded
```

```
into the File              (0-2Giga)[99188] :
```

```
(Fast_Convert NoFast_Convert RMS_Puts)
```

```
File Reloading Method      (Keyword)[Fast]      :
```

```
Number of Additional Records to be Added After
```

```
the Reloading the File      (0-2147384457)[0]      :
```

```
Key 0 Segmentation desired  (Yes/No)[No]      :
```

[illegible]

PV-Prolog Version	3	KT-Key	0	Type	String	EM-Emphasis	Flatter	(	6)
DK-Dup Key	0	Values	No	KL-Key	0	Length	12	KP-Key	0
								Position	0
RC-Data Record Comp	14%	KC-Data	Key	Comp	25%	IC-Index	Record	Comp	16%
BF-Bucket Fill	100%	RF-Record	Format	Variable	RS-Mean	Record	Size	120	
LM-Load Method	Fast	Conv	IL-Initial	Load	99188	AR-Added	Records	0	

(Type "FD" to Finish Design)

Which File Parameter (Mnemonic)[refresh] : fd

Text for FDL Title Section (1-126 chars)[null]:

Emphasis Used In Defining Default:	(	Flatter_files	)
Suggested Bucket Sizes:	(	3	6 30)
Number of Levels in Index:	(	3	2 1)
Number of Buckets in Index:	(	96	24 1)
Pages Required to Cache Index:	(	288	144 30)
Processing Used to Search Index:	(	222	298 751)
Key 0 Bucket Size	(1-63)[6]	:	

The Depth of Key 0 is Estimated to be No Greater than 2 Index levels, which is 3 Total levels.

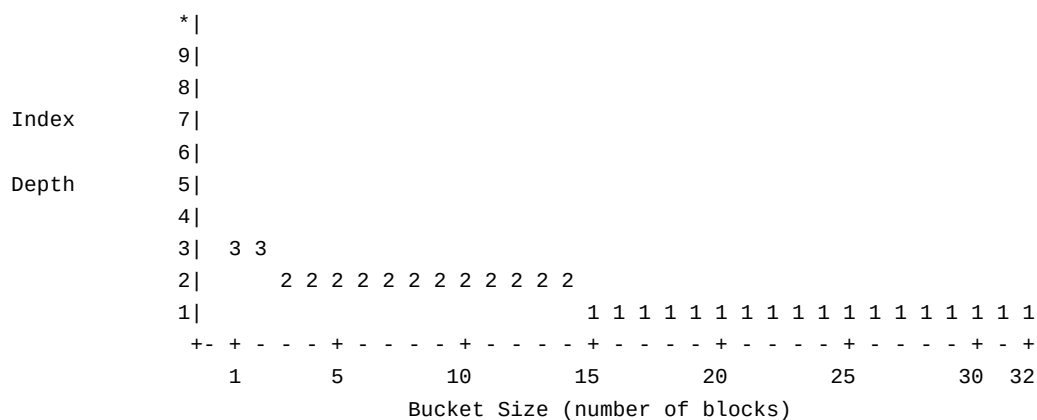
Press RETURN to continue (^Z for Main Menu)

The Definition of Key 1 will be replaced.
Press RETURN to continue (^Z for Main Menu)

(Line Fill Key)

Graph type to display (Keyword)[Line] :

Key 1 Segmentation desired (Yes/No)[No] :



PV-Prolog Version 3 KT-Key 1 Type String EM-Emphasis Flatter (15)
 DK-Dup Key 1 Values Yes KL-Key 1 Length 20 KP-Key 1 Position 12
 RC-Data Record Comp 0% KC-Data Key Comp 43% IC-Index Record Comp 26%
 BF-Bucket Fill 100% RF-Record Format Variable RS-Mean Record Size 120
 LM-Load Method Fast_Conv IL-Initial Load 99188 AR-Added Records 0
 (Type "FD" to Finish Design)
 Which File Parameter (Mnemonic)[refresh] :

```

      *|
      9|
      8|
Index  7|
      6|
Depth  5|
      4|
      3|  3 3
      2|      2 2 2 2 2 2 2 2 2 2 2 2
      1|      1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
      +-+ - - - + - - - - + - - - - + - - - - + - - - - + - - +
        1      5      10      15      20      25      30 32
        Bucket Size (number of blocks)

```

```

PV-Prolog Version      3 KT-Key  1 Type      String EM-Emphasis Flatter (15)
DK-Dup Key  1 Values  Yes KL-Key  1 Length      20 KP-Key  1 Position      12
RC-Data Record Comp    0% KC-Data Key Comp      43% IC-Index Record Comp  26%
BF-Bucket Fill          100% RF-Record Format Variable RS-Mean Record Size  120
LM-Load Method  Fast_Conv IL-Initial Load      99188 AR-Added Records      0
(Type "FD" to Finish Design)
Which File Parameter  (Mnemonic)[refresh]      : fd

```

```

Emphasis Used In Defining Default:  (    Flatter_files    )
Suggested Bucket Sizes:              (      3      15      24 )
Number of Levels in Index:            (      2      1      1 )
Number of Buckets in Index:           (     24      1      1 )
Pages Required to Cache Index:         (     72     15     24 )
Processing Used to Search Index:       (    120    303    486 )
Key  1 Bucket Size                    (1-63)[15]      :

```

The Depth of Key 1 is Estimated to be No Greater
than 1 Index levels, which is 2 Total levels.
Press RETURN to continue (^Z for Main Menu)

(Add Delete Exit Help Invoke Modify Quit Set View)
Main Editor Function (Keyword)[Help] : **exit**

_DKA300:[ELLIS.RMS]INDEXED.FDL;10 71 lines

\$ **type indexed.fdl**

IDENT "14-MAR-1999 02:49:31 OpenVMS FDL Editor"

SYSTEM

SOURCE "OpenVMS"

FILE

CONTIGUOUS no
FILE_MONITORING no
GLOBAL_BUFFER_COUNT 0
NAME "_DKA300:[ELLIS.RMS]INDEXED.DAT;1"
ORGANIZATION indexed
OWNER [ELLIS]
PROTECTION (system:RWED, owner:RWED, group:RE, world:)

RECORD

BLOCK_SPAN yes
CARRIAGE_CONTROL carriage_return
FORMAT variable
SIZE 220

AREA 0

ALLOCATION 23811
BEST_TRY_CONTIGUOUS yes
BUCKET_SIZE 6
EXTENSION 5952

AREA 1

ALLOCATION 144
BEST_TRY_CONTIGUOUS yes
BUCKET_SIZE 6
EXTENSION 36

AREA 2

ALLOCATION	4890
BEST_TRY_CONTIGUOUS	yes
BUCKET_SIZE	15
EXTENSION	1245

KEY 0

CHANGES	no
DATA_AREA	0
DATA_FILL	100
DATA_KEY_COMPRESSION	yes
DATA_RECORD_COMPRESSION	yes
DUPLICATES	no
INDEX_AREA	1
INDEX_COMPRESSION	yes
INDEX_FILL	100
LEVEL1_INDEX_AREA	1
NAME	" "
NULL_KEY	no
PROLOG	3
SEG0_LENGTH	12
SEG0_POSITION	0
TYPE	string

KEY 1

CHANGES	no
DATA_AREA	2
DATA_FILL	100
DATA_KEY_COMPRESSION	yes
DUPLICATES	yes
INDEX_AREA	2
INDEX_COMPRESSION	yes
INDEX_FILL	100

```
        LEVEL1_INDEX_AREA      2
        NAME                    ""
        NULL_KEY                no
        SEG0_LENGTH             20
        SEG0_POSITION           12
        TYPE                     string
$ rename indexed.dat indexed.old2
$
$ convert/fdl=indexed.fdl indexed.old2 indexed.dat
$
$ sh status
  Status on 14-MAR-1999 03:00:03.33      Elapsed CPU :   0 00:06:08.03
  Buff. I/O :      1209    Cur. ws. :   2000    Open files :         0
  Dir. I/O :   140102    Phys. Mem. :    896    Page Faults :    2380
$ run read_seq_idx
$ sh stat
  Status on 14-MAR-1999 03:00:57.41      Elapsed CPU :   0 00:06:49.30
  Buff. I/O :      1262    Cur. ws. :   2000    Open files :         0
  Dir. I/O :   143997    Phys. Mem. :   1024    Page Faults :    2425
$
$
$
```

```

$ ana/rms/fdl indexed.dat
$ type indexed.fdl
IDENT          "14-MAR-1999 03:02:23  OpenVMS ANALYZE/RMS_FILE Utility"

SYSTEM
    SOURCE                      OpenVMS

FILE
    ALLOCATION                    28845
    BEST_TRY_CONTIGUOUS         yes
    BUCKET_SIZE                  15
    CLUSTER_SIZE                 3
    CONTIGUOUS                   no
    EXTENSION                     5952
    FILE_MONITORING              no
    GLOBAL_BUFFER_COUNT          0
    NAME                         "_DKA300:[ELLIS.RMS]INDEXED.DAT;1"
    ORGANIZATION                 indexed
    OWNER                        [ELLIS]
    PROTECTION                   (system:RWED, owner:RWED, group:RE, world:)

RECORD
    BLOCK_SPAN                   yes
    CARRIAGE_CONTROL             carriage_return
    FORMAT                       variable
    SIZE                         220

AREA 0
    ALLOCATION                    23811
    BEST_TRY_CONTIGUOUS         yes
    BUCKET_SIZE                  6
    EXTENSION                     5952

AREA 1
    ALLOCATION                    144
    BEST_TRY_CONTIGUOUS         yes
    BUCKET_SIZE                  6
    EXTENSION                     36

```



```
AREA 2
  ALLOCATION          4890
  BEST_TRY_CONTIGUOUS yes
  BUCKET_SIZE        15
  EXTENSION          1245

KEY 0
  CHANGES            no
  DATA_KEY_COMPRESSION yes
  DATA_RECORD_COMPRESSION yes
  DATA_AREA          0
  DATA_FILL          100
  DUPLICATES          no
  INDEX_AREA          1
  INDEX_COMPRESSION   yes
  INDEX_FILL          100
  LEVEL1_INDEX_AREA   1
  NAME                ""
  NULL_KEY            no
  PROLOG              3
  SEG0_LENGTH         12
  SEG0_POSITION        0
  TYPE                string

KEY 1
  CHANGES            no
  DATA_KEY_COMPRESSION yes
  DATA_AREA          2
  DATA_FILL          100
  DUPLICATES          yes
  INDEX_AREA          2
  INDEX_COMPRESSION   yes
  INDEX_FILL          100
  LEVEL1_INDEX_AREA   2
  NAME                ""
  NULL_KEY            no
  SEG0_LENGTH         20
  SEG0_POSITION        12
  TYPE                string
```

```

ANALYSIS_OF_AREA 0
    RECLAIMED_SPACE      0

ANALYSIS_OF_AREA 1
    RECLAIMED_SPACE      0

ANALYSIS_OF_AREA 2
    RECLAIMED_SPACE      0

ANALYSIS_OF_KEY 0
    DATA_FILL           98
    DATA_KEY_COMPRESSION 29
    DATA_RECORD_COMPRESSION 14
    DATA_RECORD_COUNT    99188
    DATA_SPACE_OCCUPIED  22278
    DEPTH                 2
    INDEX_COMPRESSION     12
    INDEX_FILL            93
    INDEX_SPACE_OCCUPIED  96
    LEVEL1_RECORD_COUNT   3713
    MEAN_DATA_LENGTH      120
    MEAN_INDEX_LENGTH     14

ANALYSIS_OF_KEY 1
    DATA_FILL           99
    DATA_KEY_COMPRESSION 44
    DATA_RECORD_COUNT    98595
    DATA_SPACE_OCCUPIED  3915
    DEPTH                 1
    DUPLICATES_PER_SIDR   0
    INDEX_COMPRESSION     24
    INDEX_FILL            56
    INDEX_SPACE_OCCUPIED  15
    LEVEL1_RECORD_COUNT   261
    MEAN_DATA_LENGTH      20
    MEAN_INDEX_LENGTH     22

$

```

Notes on Optimizations

- Performance metrics on optimized file (accessed sequentially):
 - Elapsed time: :54.08 (vs. 1:22.20)
 - Direct I/Os 3895 (vs. 12102)



Note

The larger bucket size yields larger I/Os for sequential access.

- For a more accurate analysis, you must also test random access.
- The following tests illustrate performance metrics for random reads of 20,000 records.

Single Test (Random Reads)	Elapsed Time	DIOs
Random populated file	6:08.65	27204
Converted file	4:31.03	21175
Optimized file	4:16.31	17953

Example Testing Random Access of Indexed Files

```
$
$
$ @rand_test
$!Single access random (20000)
$ rename indexed.old1 indexed.dat
$ show status
  Status on 15-MAR-1999 14:56:55.13      Elapsed CPU : 0 00:01:41.52
  Buff. I/O :      189      Cur. ws. : 2000      Open files :      1
  Dir. I/O :     32265      Phys. Mem. : 1056      Page Faults :     419
$ run rand_idx
$ show status
  Status on 15-MAR-1999 15:03:03.78      Elapsed CPU : 0 00:02:26.63
  Buff. I/O :      205      Cur. ws. : 2000      Open files :      1
  Dir. I/O :     59469      Phys. Mem. : 960      Page Faults :     517
$ rename indexed.dat indexed.old1
$ rename indexed.old2 indexed.dat
$ show status
  Status on 15-MAR-1999 15:03:04.07      Elapsed CPU : 0 00:02:26.72
  Buff. I/O :      227      Cur. ws. : 2000      Open files :      1
  Dir. I/O :     59477      Phys. Mem. : 944      Page Faults :     586
$ run rand_idx
$ show status
  Status on 15-MAR-1999 15:07:35.10      Elapsed CPU : 0 00:02:57.70
  Buff. I/O :      243      Cur. ws. : 2000      Open files :      1
  Dir. I/O :     80652      Phys. Mem. : 960      Page Faults :     662
$ rename indexed.dat indexed.old2
$ rename indexed.old3 indexed.dat
$ show status
  Status on 15-MAR-1999 15:07:35.41      Elapsed CPU : 0 00:02:57.80
  Buff. I/O :      265      Cur. ws. : 2000      Open files :      1
  Dir. I/O :     80660      Phys. Mem. : 928      Page Faults :     728
$ run rand_idx
$ show status
  Status on 15-MAR-1999 15:11:51.72      Elapsed CPU : 0 00:03:26.19
  Buff. I/O :      281      Cur. ws. : 2000      Open files :      1
  Dir. I/O :     98613      Phys. Mem. : 928      Page Faults :     816
$ rename indexed.dat indexed.old3
```

RMS I/O Buffering Considerations

Module 19

Module Outline

Objectives

This module describes the use of I/O buffering for the reduction of I/O operations and enhancement of file-related performance.

Topics

- Buffers and Buckets
- Read Ahead
- Write Behind
- Deferred Write
- Setting Up Multiple Buffers
- Determining the Number of Local Buffers

Buffers and Buckets

Sequential Files

- I/O is performed in blocks with the number of blocks per I/O being defined by multiblock count.
- Data for I/O is stored in memory by RMS in buffers stored in the process P1 space.
- Reads can complete by copying data from the P1 space buffer to the user buffer, if the record has already been read.

Relative Files

- I/O is performed in units of buckets where the bucket size is characteristic of the file.
- Data for I/O is stored in buffers.

Indexed Files

- I/O is performed in units of buckets.
- Buckets correspond to buckets in the indexed file structure, containing indexes or data.
- I/O is stored in buffers.
- Buffers are allocated large enough to hold the largest bucket.



Note

With areas, you can define different bucket sizes for indexes and data buckets.

Design Considerations

- Preallocate space on the disk, if possible.
- Consider the size, number, and type of RMS buffers.
 - A few large buffers for sequential files.
 - Several small buffers for indexed files being accessed randomly.
- Consider the window size, if the file is not contiguous. Avoid window turns.
- If the file is shared, consider Global buffers.

Multiblock Count Example

```

$ set verify
$ @seq_buff
$ define seq big.seq
$ show rms

```

	MULTI- BLOCK COUNT	Indexed	Relative	MULTIBUFFER COUNTS				NETWORK BLOCK COUNT
				Disk	Magtape	Unit Record		
Process	0	0	0	0	0	0	0	
System	16	0	0	0	0	0	8	

```

$ show status

```

	Prolog	Extend	Quantity
Process	0		0
System	0		0

```

$ show status
Status on 2-APR-1999 21:38:38.00 Elapsed CPU : 0 00:00:00.26
Buff. I/O : 63 Cur. ws. : 2000 Open files : 1
Dir. I/O : 24 Phys. Mem. : 1184 Page Faults : 138
$ run write_seq
$ show status
Status on 2-APR-1999 21:44:49.58 Elapsed CPU : 0 00:01:08.54
Buff. I/O : 4326 Cur. ws. : 2000 Open files : 1
Dir. I/O : 18554 Phys. Mem. : 1168 Page Faults : 187
$ run read_seq
$ show status
Status on 2-APR-1999 21:45:47.19 Elapsed CPU : 0 00:01:47.64
Buff. I/O : 4340 Cur. ws. : 2000 Open files : 1
Dir. I/O : 28567 Phys. Mem. : 1152 Page Faults : 229
$ delete big.seq;
$ set rms/block=64
$ show status
Status on 2-APR-1999 21:45:47.59 Elapsed CPU : 0 00:01:47.70
Buff. I/O : 4356 Cur. ws. : 2000 Open files : 1
Dir. I/O : 28583 Phys. Mem. : 1024 Page Faults : 288
$ run write_seq
$ show status
Status on 2-APR-1999 21:48:54.85 Elapsed CPU : 0 00:02:35.97
Buff. I/O : 5458 Cur. ws. : 2000 Open files : 1
Dir. I/O : 33322 Phys. Mem. : 928 Page Faults : 359
$ run read_seq
$ show status
Status on 2-APR-1999 21:49:44.20 Elapsed CPU : 0 00:03:10.90
Buff. I/O : 5472 Cur. ws. : 2000 Open files : 1
Dir. I/O : 35827 Phys. Mem. : 912 Page Faults : 409
$ delete big.seq;

```



```

$ set rms/block=127
$ show status
  Status on   2-APR-1999 21:49:44.61      Elapsed CPU :   0 00:03:11.01
  Buff. I/O :    5488    Cur. ws. :   2000    Open files :      1
  Dir. I/O :   35847    Phys. Mem. :    896    Page Faults :    487
$ run write_seq
$ show status
  Status on   2-APR-1999 21:52:12.17      Elapsed CPU :   0 00:03:55.37
  Buff. I/O :    6053    Cur. ws. :   2000    Open files :      1
  Dir. I/O :   38229    Phys. Mem. :    928    Page Faults :    555
$ run read_seq
$ show status
  Status on   2-APR-1999 21:53:01.25      Elapsed CPU :   0 00:04:30.09
  Buff. I/O :    6067    Cur. ws. :   2000    Open files :      1
  Dir. I/O :   39491    Phys. Mem. :    864    Page Faults :    618
$ delete big.seq

```

Notes on Multiblock Count Example

- A sequential file of 1,000,000 records was created and then read in each test.
- The disk being processed was an RZ26L.

Multiblock Size	Elapsed Time	Direct I/Os
16 (write)	6:19.58	18534
64 (write)	3:07.26	4739
127 (write)	2:28.56	2382
16 (read)	:57.21	10013
64 (read)	:49.35	2505
127 (read)	:48.08	1272

Read Ahead/Write Behind

- Apply only to nonshared sequential files.
 - Read Ahead can be enabled by setting the **RAB\$M_RAH** mask in **RAB\$L_ROP**.
 - Write Behind can be enabled by setting the **RAB\$M_WBH** mask in **RAB\$L_ROP**.

Read Ahead

Processing allows users to take advantage of posting multiple I/Os to disk asynchronously and processing data from beginning blocks while pending blocks are still being read.

Write Behind

- Allows the user program to write records into the data buffer until the buffer is full prior to initiating write of buffer.
- While I/O is in progress on the initial buffer, process can start writing to the next buffer.
- Buffers will be flushed to disk when the file is closed.
- Deferred writing could result in loss of integrity of the file if the system crashes.
- To force a write of the buffer, call **SYSS\$FLUSH** or a high-level language equivalent.

Deferred Write

Deferred write applies to relative, indexed, and shared sequential files.

- Can be enabled by setting **FAB\$M_DFW** in field **FAB\$L_FOP**.
- Allows process to write buffer until the buffer is needed for another RMS bucket read.
- Can minimize I/O operations for writes.
- Option trades off write I/O performance rather than the potential data loss during a system crash.
- Data is forced back to disk when a call is made to either **SYS\$CLOSE** or **SYS\$FLUSH**.



Note

Read access to buffers from a file is always enabled, because there is no data integrity issue, and processes will always benefit from minimizing I/O operations.

Setting Up Multiple Buffers

- Multiple buffers allow processes to cache multiple buckets in memory.
 - When processes go back to buckets previously processed, you do not have to perform an I/O operation if the buffer has not been lost from memory.
 - There is a high likelihood of reprocessing buckets in the case of indexed files, due to accessing of index buckets.
- Multiple buffers can be established by setting the **RAB\$B_MBF** field in the record access block to contain the desired number of buffers.
 - High-level languages allow the establishment of multiple buffers with following options on file open:

Language	Option
BASIC	BUFFER n
C/C++	Requires the user to access the RAB directly, or "mbf = n" on creat, fopen, or open.
COBOL	RESERVE n
FORTRAN	BUFFERCOUNT=n
PASCAL	Requires that the program use the USERACTION routine to access RAB (RAB\$W_MBF=n)
PL/I	ENVIRONMENT (...MULTIBUFFER_COUNT(n)...)

- If program options have not been set, check process defaults.
 - Process defaults can be set by using:
`$SET RMS_DEFAULT/type/BUFFER=n`
where:
`/type = /INDEX, /RELATIVE, or /SEQUENTIAL`
- If no process default is set and there are no program settings, use system defaults.
 - System defaults can be established by using:
`$SET RMS_DEFAULT/type/SYSTEM/BUFFER=n`
where:
`/type = /INDEX, /RELATIVE, or /SEQUENTIAL`
- This is equivalent to setting the following **SYSGEN** parameters:
 - **RMS_DFMBCREL**—for relative files.
 - **RMS_DFMBCIDX**—for indexed files.
 - **RMS_DFMBCSDK**—for sequential disk access.

**Note**

Changing system defaults requires **CMKRNL** privilege.

Determining the Number of Local Buffers

Sequential Files

- Generally, there is no advantage to more than one buffer if Read Ahead and Write Behind are disabled.
- If disabled, you can optimize I/O by issuing larger transfers with larger multiblock count (**\$SET RMS/BLOCK=*n***).
- Minimizes seeking and latency characteristics of disk I/O.
- If Read Ahead (**RAH**) and/or Write Behind (**WBH**) is enabled, consider multiple buffers.
- Remember—trading off pages in the working set for fewer or concurrent I/Os.

Relative Files

- Unless reaccessing records, more buffers are not a big advantage because of the generally random nature of relative file access.
- If access is localized through application design, consider more buffers.

Indexed Files

- Buffers are weighted based on the index level of a bucket stored in the buffer.
- When buffers are reused, a bucket's lowest weight buffer is discarded so that the buffer can be reused.
- Therefore, index-level buckets tend to be cached over data buckets.
- For random processing, you should attempt to cache as many index buckets as is reasonable under memory constraints.
- For purely sequential access, more buffers provide no gain.

Local Buffer Example (Caching Disabled)

```

$ set verify
$ @buff_test
$ set rms/buffer=0/indexed
$ show memory/cache/full
      System Memory Resources on 1-APR-1999 22:25:10.89
The Virtual I/O Cache is DISABLED on this node.
$show status
  Status on 1-APR-1999 22:25:10.91 Elapsed CPU : 0 00:00:11.91
  Buff. I/O : 214 Cur. ws. : 2000 Open files : 1
  Dir. I/O : 10811 Phys. Mem. : 880 Page Faults : 1052
$run rand_idx
$show status
  Status on 1-APR-1999 22:31:29.40 Elapsed CPU : 0 00:00:43.59
  Buff. I/O : 230 Cur. ws. : 2000 Open files : 1
  Dir. I/O : 46779 Phys. Mem. : 976 Page Faults : 1130
$show memory/cache/full
      System Memory Resources on 1-APR-1999 22:31:29.52
The Virtual I/O Cache is DISABLED on this node.
$set rms/buffer=8/indexed
$show memory/cache/full
      System Memory Resources on 1-APR-1999 22:31:29.75
The Virtual I/O Cache is DISABLED on this node.
$show status
  Status on 1-APR-1999 22:31:29.77 Elapsed CPU : 0 00:00:43.69
  Buff. I/O : 255 Cur. ws. : 2000 Open files : 1
  Dir. I/O : 46779 Phys. Mem. : 912 Page Faults : 1253
$run rand_idx
$show status
  Status on 1-APR-1999 22:37:34.47 Elapsed CPU : 0 00:01:12.45
  Buff. I/O : 271 Cur. ws. : 2000 Open files : 1
  Dir. I/O : 78767 Phys. Mem. : 960 Page Faults : 1333
$show memory/cache/full
      System Memory Resources on 1-APR-1999 22:37:34.59
The Virtual I/O Cache is DISABLED on this node.
$set rms/buffer=32/indexed
$show memory/cache/full
      System Memory Resources on 1-APR-1999 22:37:34.82
The Virtual I/O Cache is DISABLED on this node.
$show status
  Status on 1-APR-1999 22:37:34.84 Elapsed CPU : 0 00:01:12.54
  Buff. I/O : 296 Cur. ws. : 2000 Open files : 1
  Dir. I/O : 78767 Phys. Mem. : 944 Page Faults : 1457
$run rand_idx
$show status
  Status on 1-APR-1999 22:42:11.64 Elapsed CPU : 0 00:01:34.69
  Buff. I/O : 312 Cur. ws. : 2000 Open files : 1
  Dir. I/O : 98744 Phys. Mem. : 960 Page Faults : 1561
$show memory/cache/full
      System Memory Resources on 1-APR-1999 22:42:11.76
The Virtual I/O Cache is DISABLED on this node.

```

```
$set rms/buffer=100/indexed
$show memory/cache/full
      System Memory Resources on 1-APR-1999 22:42:11.99
The Virtual I/O Cache is DISABLED on this node.
$show status
  Status on 1-APR-1999 22:42:12.01    Elapsed CPU : 0 00:01:34.79
  Buff. I/O :      337    Cur. ws. : 2000    Open files :      1
  Dir. I/O :     98744    Phys. Mem. :   944    Page Faults :    1685
$run rand_idx
$show status
  Status on 1-APR-1999 22:46:44.74    Elapsed CPU : 0 00:01:58.16
  Buff. I/O :      353    Cur. ws. : 2000    Open files :      1
  Dir. I/O :    118360    Phys. Mem. :   864    Page Faults :    1877
$show memory/cache/full
      System Memory Resources on 1-APR-1999 22:46:44.85
The Virtual I/O Cache is DISABLED on this node.
$ set noverify
$
```

Notes on the Local Buffer Example (Caching Disabled)

- These tests were run on a file with approximately 100,000 records.
- 20,000 random records were read in each test.
- Caching was disabled to illustrate the effects of buffering, in elimination of I/O.

Number of Buffers	Elapsed Time	Direct I/Os
0 (2 default)	6:18.49	35968
8	6:04.70	31988
32	4:76.84	19977
100	4:32.73	19616

Local Buffer Example (Caching Enabled)

```

$ set verify
$ @buff_test
$!Single access random (20000)
$set rms/buffer=0/indexed
$show memory/cache/full
      System Memory Resources on 1-APR-1999 23:05:37.92
Virtual I/O Cache
  Total Size (Kbytes)          3200    Read IO Count          15275
  Free Kbytes                   0      Read Hit Count          4932
  Kbytes in Use                 3200    Read Hit Rate           32%
  Write IO Bypassing Cache      0      Write IO Count           40
  Files Retained                100    Read IO Bypassing Cache  714
$show status
  Status on 1-APR-1999 23:05:37.94    Elapsed CPU : 0 00:00:11.86
  Buff. I/O : 254    Cur. ws. : 2000    Open files : 1
  Dir. I/O : 7283    Phys. Mem. : 896    Page Faults : 966
$run rand_idx
$show status
  Status on 1-APR-1999 23:09:54.48    Elapsed CPU : 0 00:00:40.93
  Buff. I/O : 270    Cur. ws. : 2000    Open files : 1
  Dir. I/O : 25222    Phys. Mem. : 976    Page Faults : 1047
$show memory/cache/full
      System Memory Resources on 1-APR-1999 23:09:54.60
Virtual I/O Cache
  Total Size (Kbytes)          3200    Read IO Count          51258
  Free Kbytes                   0      Read Hit Count          22963
  Kbytes in Use                 3200    Read Hit Rate           44%
  Write IO Bypassing Cache      0      Write IO Count           40
  Files Retained                100    Read IO Bypassing Cache  715
$set rms/buffer=8/indexed
$show memory/cache/full
      System Memory Resources on 1-APR-1999 23:09:54.75
Virtual I/O Cache
  Total Size (Kbytes)          3200    Read IO Count          51272
  Free Kbytes                   0      Read Hit Count          22966
  Kbytes in Use                 3200    Read Hit Rate           44%
  Write IO Bypassing Cache      0      Write IO Count           40
  Files Retained                100    Read IO Bypassing Cache  717
$show status
  Status on 1-APR-1999 23:09:54.80    Elapsed CPU : 0 00:00:41.00
  Buff. I/O : 305    Cur. ws. : 2000    Open files : 1
  Dir. I/O : 25222    Phys. Mem. : 944    Page Faults : 1161
$run rand_idx
$show status
  Status on 1-APR-1999 23:14:09.80    Elapsed CPU : 0 00:01:09.48
  Buff. I/O : 321    Cur. ws. : 2000    Open files : 1
  Dir. I/O : 43139    Phys. Mem. : 976    Page Faults : 1248

```

```
$show memory/cache/full
      System Memory Resources on 1-APR-1999 23:14:09.92
Virtual I/O Cache
  Total Size (Kbytes)      3200    Read IO Count      83278
  Free Kbytes              0       Read Hit Count     37038
  Kbytes in Use            3200    Read Hit Rate      44%
  Write IO Bypassing Cache 0       Write IO Count     42
  Files Retained           100     Read IO Bypassing Cache 718
$set rms/buffer=32/indexed
$show memory/cache/full
      System Memory Resources on 1-APR-1999 23:14:10.07
Virtual I/O Cache
  Total Size (Kbytes)      3200    Read IO Count      83292
  Free Kbytes              0       Read Hit Count     37041
  Kbytes in Use            3200    Read Hit Rate      44%
  Write IO Bypassing Cache 0       Write IO Count     42
  Files Retained           100     Read IO Bypassing Cache 720
$show status
  Status on 1-APR-1999 23:14:10.12    Elapsed CPU : 0 00:01:09.61
  Buff. I/O : 356    Cur. ws. : 2000    Open files : 1
  Dir. I/O : 43139    Phys. Mem. : 944    Page Faults : 1369
$run rand_idx
$show status
  Status on 1-APR-1999 23:18:21.52    Elapsed CPU : 0 00:01:33.20
  Buff. I/O : 372    Cur. ws. : 2000    Open files : 1
  Dir. I/O : 61002    Phys. Mem. : 976    Page Faults : 1481
$show memory/cache/full
      System Memory Resources on 1-APR-1999 23:18:21.64
Virtual I/O Cache
  Total Size (Kbytes)      3200    Read IO Count      103284
  Free Kbytes              0       Read Hit Count     39156
  Kbytes in Use            3200    Read Hit Rate      37%
  Write IO Bypassing Cache 0       Write IO Count     42
  Files Retained           100     Read IO Bypassing Cache 721
$set rms/buffer=100/indexed
$show memory/cache/full
      System Memory Resources on 1-APR-1999 23:18:21.81
Virtual I/O Cache
  Total Size (Kbytes)      3200    Read IO Count      103298
  Free Kbytes              0       Read Hit Count     39159
  Kbytes in Use            3200    Read Hit Rate      37%
  Write IO Bypassing Cache 0       Write IO Count     42
  Files Retained           100     Read IO Bypassing Cache 723
$show status
  Status on 1-APR-1999 23:18:21.85    Elapsed CPU : 0 00:01:33.32
  Buff. I/O : 407    Cur. ws. : 2000    Open files : 1
  Dir. I/O : 61002    Phys. Mem. : 944    Page Faults : 1596
$run rand_idx
$show status
  Status on 1-APR-1999 23:22:33.48    Elapsed CPU : 0 00:01:59.04
  Buff. I/O : 423    Cur. ws. : 2000    Open files : 1
  Dir. I/O : 78874    Phys. Mem. : 896    Page Faults : 1784
```

```

$show memory/cache/full
      System Memory Resources on  1-APR-1999 23:22:33.60
Virtual I/O Cache
  Total Size (Kbytes)           3200   Read IO Count           122930
  Free Kbytes                   0      Read Hit Count           40904
  Kbytes in Use                 3200   Read Hit Rate             33%
  Write IO Bypassing Cache      0      Write IO Count            43
  Files Retained                100   Read IO Bypassing Cache   724
$ set noverify

```

Notes on the Local Buffer Example (Caching Enabled)

- These examples were generated with the virtual I/O cache enabled to illustrate the interaction between the virtual I/O cache and RMS buffering.
- In stand-alone tests, the impact of RMS buffering is minimized by the Virtual I/O cache.
- In a multiuser, multifile access environment, the impact of buffering will be more noticeable.
- As the number of buffers increases, the cache dependencies decrease.

Number of Buffers	Elapsed Time	Direct I/Os	Cache Hits
0 (2 default)	4:16.54	17939	18031
8	4:15.00	17917	14372
32	4:11.40	17863	2175
100	4:11.63	17872	1745

Global Buffers

- Allow for sharing of data records in memory between processes accessing the same file.
 - RMS global buffers are implemented as page file global sections.
 - Offer benefits similar to local buffers with the added advantage that another process may have previously read the bucket of interest, eliminating any necessary I/O for process.
- RMS global buffers can be established on a file by using the following options:
 - Storing the desired number of buffers in the field **FAB\$W_GBC** prior to creation of the file.
 - Using FDL, **FILE** section, and **GLOBAL_BUFFER_COUNT=*n*** secondary attribute.
`$SET FILE/GLOBAL=n file.`
- Global buffers operate similar to local buffers in terms of discarding buffers, with the exception that scan to discard a buffer is minimized to checking eight buffers.
- Deferred write can cause additional overhead due to locking requirements and copying of the global buffer into process' local buffer.
- Attempt to keep files, with global buffers enabled, open as long as possible, because closing the file with global buffers enabled forces modified page writing (more of an issue in pre-V5.0).
- Be cautious of locking overhead in an OpenVMS Cluster.

Global Buffers Example

```

$ set verify
$ @glob_test
$set file/glob=0 indexed.dat
$show memory/cache/full
      System Memory Resources on 2-APR-1999 22:44:28.77
Virtual I/O Cache
  Total Size (Kbytes)      3200    Read IO Count      374952
  Free Kbytes              0      Read Hit Count    166821
  Kbytes in Use            3200    Read Hit Rate     44%
  Write IO Bypassing Cache 5957    Write IO Count    67377
  Files Retained           92      Read IO Bypassing Cache 9917
$spawn/nowait run rand_idx
%DCL-S-SPAWNED, process ELLIS_1 spawned
$spawn/nowait run rand_idx
%DCL-S-SPAWNED, process ELLIS_2 spawned
$spawn/nowait run rand_idx
%DCL-S-SPAWNED, process ELLIS_3 spawned
$show status
  Status on 2-APR-1999 22:44:30.14    Elapsed CPU : 0 00:00:00.33
  Buff. I/O : 123    Cur. ws. : 2000    Open files : 5
  Dir. I/O : 19     Phys. Mem. : 1104    Page Faults : 210
$run rand_idx
$show status
  Status on 2-APR-1999 22:49:43.99    Elapsed CPU : 0 00:00:37.24
  Buff. I/O : 143    Cur. ws. : 2000    Open files : 2
  Dir. I/O : 4428    Phys. Mem. : 1008    Page Faults : 308
$wait ::30.0
$show memory/cache/full
      System Memory Resources on 2-APR-1999 22:50:14.11
Virtual I/O Cache
  Total Size (Kbytes)      3200    Read IO Count      519405
  Free Kbytes              0      Read Hit Count    293288
  Kbytes in Use            3200    Read Hit Rate     56%
  Write IO Bypassing Cache 5957    Write IO Count    67379
  Files Retained           92      Read IO Bypassing Cache 9924
$
$set file/glob=20 indexed.dat
$show memory/cache/full
      System Memory Resources on 2-APR-1999 22:50:14.33
Virtual I/O Cache
  Total Size (Kbytes)      3200    Read IO Count      519420
  Free Kbytes              0      Read Hit Count    293291
  Kbytes in Use            3200    Read Hit Rate     56%
  Write IO Bypassing Cache 5957    Write IO Count    67379
  Files Retained           92      Read IO Bypassing Cache 9926
$spawn/nowait run rand_idx
%DCL-S-SPAWNED, process ELLIS_1 spawned
$spawn/nowait run rand_idx
%DCL-S-SPAWNED, process ELLIS_2 spawned
$spawn/nowait run rand_idx
%DCL-S-SPAWNED, process ELLIS_3 spawned

```

```
$show status
  Status on    2-APR-1999 22:50:15.63      Elapsed CPU :    0 00:00:37.35
  Buff. I/O :    237      Cur. ws. :    2000      Open files :        5
  Dir. I/O :    4434      Phys. Mem. :    992      Page Faults :       456

$run rand_idx
$show status
  Status on    2-APR-1999 22:55:11.61      Elapsed CPU :    0 00:01:08.19
  Buff. I/O :    259      Cur. ws. :    2000      Open files :        1
  Dir. I/O :    8872      Phys. Mem. :   1008      Page Faults :       569

$wait ::30.0
$show memory/cache/full
      System Memory Resources on 2-APR-1999 22:55:41.73
Virtual I/O Cache
  Total Size (Kbytes)          3200      Read IO Count          547171
  Free Kbytes                   0      Read Hit Count          303044
  Kbytes in Use                 3200      Read Hit Rate             55%
  Write IO Bypassing Cache     5957      Write IO Count            67382
  Files Retained                92      Read IO Bypassing Cache   9952

$
$set file/glob=100 indexed.dat
$show memory/cache/full
      System Memory Resources on 2-APR-1999 22:55:41.95
Virtual I/O Cache
  Total Size (Kbytes)          3200      Read IO Count          547186
  Free Kbytes                   0      Read Hit Count          303047
  Kbytes in Use                 3200      Read Hit Rate             55%
  Write IO Bypassing Cache     5957      Write IO Count            67382
  Files Retained                92      Read IO Bypassing Cache   9954

$spawn/nowait run rand_idx
%DCL-S-SPAWNED, process ELLIS_1 spawned
$spawn/nowait run rand_idx
%DCL-S-SPAWNED, process ELLIS_2 spawned
$spawn/nowait run rand_idx
%DCL-S-SPAWNED, process ELLIS_3 spawned
$show status
  Status on    2-APR-1999 22:55:43.14      Elapsed CPU :    0 00:01:08.38
  Buff. I/O :    351      Cur. ws. :    2000      Open files :        4
  Dir. I/O :    8878      Phys. Mem. :   1040      Page Faults :       712

$run rand_idx
$show status
  Status on    2-APR-1999 23:00:36.50      Elapsed CPU :    0 00:01:40.44
  Buff. I/O :    371      Cur. ws. :    2000      Open files :        3
  Dir. I/O :   13318      Phys. Mem. :   1008      Page Faults :       901

$wait ::30.0
$show memory/cache/full
      System Memory Resources on 2-APR-1999 23:01:06.62
Virtual I/O Cache
  Total Size (Kbytes)          3200      Read IO Count          568162
  Free Kbytes                   0      Read Hit Count          306088
  Kbytes in Use                 3200      Read Hit Rate             53%
  Write IO Bypassing Cache     5957      Write IO Count            67384
  Files Retained                92      Read IO Bypassing Cache   9961
```

```

$set file/glob=500 indexed.dat
$show memory/cache/full
      System Memory Resources on  2-APR-1999 23:01:06.84
Virtual I/O Cache
  Total Size (Kbytes)      3200    Read IO Count      568177
  Free Kbytes              0      Read Hit Count     306091
  Kbytes in Use            3200    Read Hit Rate      53%
  Write IO Bypassing Cache 5957    Write IO Count     67384
  Files Retained           92      Read IO Bypassing Cache 9963
$spawn/nowait run rand_idx
%DCL-S-SPAWNED, process ELLIS_1 spawned
$spawn/nowait run rand_idx
%DCL-S-SPAWNED, process ELLIS_2 spawned
$spawn/nowait run rand_idx
%DCL-S-SPAWNED, process ELLIS_3 spawned
$show status
  Status on  2-APR-1999 23:01:07.82    Elapsed CPU :  0 00:01:40.63
  Buff. I/O :    465    Cur. ws. :  2000    Open files :    3
  Dir. I/O :   13324    Phys. Mem. :  1072    Page Faults :   1018
$run rand_idx
$show status
  Status on  2-APR-1999 23:06:01.92    Elapsed CPU :  0 00:02:19.26
  Buff. I/O :    485    Cur. ws. :  2000    Open files :    1
  Dir. I/O :   17646    Phys. Mem. :   928    Page Faults :   1590
$wait ::30.0
$show memory/cache/full
      System Memory Resources on  2-APR-1999 23:06:32.03
Virtual I/O Cache
  Total Size (Kbytes)      3200    Read IO Count      585933
  Free Kbytes              0      Read Hit Count     306469
  Kbytes in Use            3200    Read Hit Rate      52%
  Write IO Bypassing Cache 5957    Write IO Count     67387
  Files Retained           92      Read IO Bypassing Cache 9970
$ set noverify

```

Notes on the Global Buffers Example

- Four processes, reading 20,000 random records, were generated to take advantage of sharing global buffers.
- Once again, the Virtual I/O cache minimized the impact of global buffers in these tests.
- The cache was affected by all four processes.

Number of Global Buffers	Elapsed Time	Direct I/Os	Cache Hits
0	5:13.85	4409	126467
20	4:55.98	4438	9753
100	4:53.36	4440	3041
500	4:54.10	4322	378

Module Outline

Objectives

This module illustrates the use of MONITOR RMS in file tuning and performance analysis.

Topics

- MONITOR RMS

MONITOR RMS

\$MONITOR RMS

To track activity on specific files, first enable file statistics by entering the DCL command:

```
$SET FILE/STATISTICS file-spec
```

- Enables accessible recording of file statistics in the global section.
- Minimal cost for enabling statistics on a file.
- Cost is only in CPU time and is significantly less than 1 percent of the normal CPU time required to access the file.

- Once statistics are enabled, enter the command:

```
$MONITOR RMS/FILE=(file1[,file2[,...]])/ITEM=(item_list)
```

- The valid items include:

- **OPERATIONS**

Displays the number and type of RMS requests (for example, sequential GETs and keyed GETs) being performed on a file.

Important when you are interested in the operation types being performed on a file accessed by an application that you did not design.

- **DATA_RATES**

Shows the amount of information processed per operation.

Of most interest when you are trying to determine activity on a file being accessed by an application that you did not design.

Locking

- Identifies locks being requested on a file, as well as bucket splits occurring on a file.
- Locks being generated are probably of interest if the file being accessed is for shared read/write access, but is only being read.
- You may want to change open to process the file as read-only.
- You will eliminate unnecessary locking overhead.
- A noticeable bucket split rate indicates that it is the time to use the **\$CONVERT** utility to reorganize the file.

Caching

- Display identifies the local and global buffer use.
- Probably the most effective item from tuning standpoint.
- Goal is to minimize I/O.
- Effective local and global buffer cache hit rates indicate eliminating I/Os.
- To adjust local buffers, if the application has not specified them, use **\$SET/RMS BUFFER=*n*/*file-org*** (where ***file-org*** can be **SEQUENTIAL**, **RELATIVE**, or **INDEXED**).
- To modify the number of global buffers on a file, can use the DCL command **\$SET FILE/GLOBAL=*n*** file specification.

All

- Shows all of the items.

MONITOR RMS Example

```
$ set file/statistics dka300:[ellis.rms]indexed.dat
$ spawn/nowait r rand_idx
$ monitor rms/file=dka300:[ellis.rms]indexed.dat/item=all
```

```
OpenVMS Monitor Utility
RMS FILE OPERATIONS
on node ELLIS
4-APR-1999 02:25:01.42
(Index) _ELLIS$DKA300:[ELLIS.RMS]INDEXED.DAT;1
Active Streams: 1
```

	CUR	AVE	MIN	MAX
\$GET Call Rate (Seq)	0.00	0.00	0.00	0.00
(Key)	72.66	72.66	72.66	72.66
(RFA)	0.00	0.00	0.00	0.00
\$FIND Call Rate (Seq)	0.00	0.00	0.00	0.00
(Key)	0.00	0.00	0.00	0.00
(RFA)	0.00	0.00	0.00	0.00
\$PUT Call Rate (Seq)	0.00	0.00	0.00	0.00
(Key)	0.00	0.00	0.00	0.00
\$READ Call Rate	0.00	0.00	0.00	0.00
\$WRITE Call Rate	0.00	0.00	0.00	0.00
\$UPDATE Call Rate	0.00	0.00	0.00	0.00
\$DELETE Call Rate	0.00	0.00	0.00	0.00
\$TRUNCATE Call Rate	0.00	0.00	0.00	0.00
\$EXTEND Call Rate	0.00	0.00	0.00	0.00
\$FLUSH Call Rate	0.00	0.00	0.00	0.00

```
OpenVMS Monitor Utility
RMS DATA RATE STATISTICS
on node ELLIS
4-APR-1999 02:25:04.43
(Index) _ELLIS$DKA300:[ELLIS.RMS]INDEXED.DAT;1
Active Streams: 1
```

	CUR	AVE	MIN	MAX
Total \$GET Call Rate	71.66	72.16	71.66	72.66
Bytes Per \$GET	120.00	120.00	120.00	120.00
Total \$PUT Call Rate	0.00	0.00	0.00	0.00
Bytes Per \$PUT	0.00	0.00	0.00	0.00
\$UPDATE Call Rate	0.00	0.00	0.00	0.00
Bytes Per \$UPDATE	0.00	0.00	0.00	0.00
\$READ Call Rate	0.00	0.00	0.00	0.00
Bytes Per \$READ	0.00	0.00	0.00	0.00
\$WRITE Call Rate	0.00	0.00	0.00	0.00
Bytes Per \$WRITE	0.00	0.00	0.00	0.00
\$TRUNCATE Call Rate	0.00	0.00	0.00	0.00
Blocks Per \$TRUNCATE	0.00	0.00	0.00	0.00
\$EXTEND Call Rate	0.00	0.00	0.00	0.00
Blocks Per \$EXTEND	0.00	0.00	0.00	0.00

```

OpenVMS Monitor Utility
RMS FILE LOCKING STATISTICS
on node ELLIS
4-APR-1999 02:25:07.43
(Index) _ELLIS$DKA300:[ELLIS.RMS]INDEXED.DAT;1
Active Streams: 1          CUR          AVE          MIN          MAX

New ENQ Rate          179.00      194.67      179.00      203.33

DEQ Rate              179.00      194.67      179.00      203.33

Converted ENQ Rate    266.66      290.12      266.66      305.33

Blocking AST Rate     0.00        0.00        0.00        0.00

Bucket Split Rate     0.00        0.00        0.00        0.00

Multi Bucket Split Rate 0.00        0.00        0.00        0.00

```

```

OpenVMS Monitor Utility
RMS CACHE STATISTICS
on node ELLIS
4-APR-1999 02:25:10.43
(Index) _ELLIS$DKA300:[ELLIS.RMS]INDEXED.DAT;1
Active Streams: 1          CUR          AVE          MIN          MAX

Local Cache Hit Percent 24.00      24.70      24.00      25.00

Local Cache Attempt Rate 349.66     337.30     306.33     349.66

Global Cache Hit Percent 0.00        0.00        0.00        0.00

Global Cache Attempt Rate 0.00        0.00        0.00        0.00

Global Buf Read I/O Rate 0.00        0.00        0.00        0.00

Global Buf Write I/O Rate 0.00        0.00        0.00        0.00

Local Buf Read I/O Rate 131.66     126.97     115.33     131.66

Local Buf Write I/O Rate 0.00        0.00        0.00        0.00

```

```
$ set file/global=200 dka300:[ellis.rms]indexed.dat
$ spawn/nowait r rand_idx
$ monitor rms/file=dka300:[ellis.rms]indexed.dat/item=all
```

OpenVMS Monitor Utility

RMS FILE OPERATIONS

on node ELLIS

4-APR-1999 02:27:29.15

(Index) _ELLIS\$DKA300:[ELLIS.RMS]INDEXED.DAT;1

Active Streams:	1	CUR	AVE	MIN	MAX
\$GET Call Rate	(Seq)	0.00	0.00	0.00	0.00
	(Key)	76.33	76.33	76.33	76.33
	(RFA)	0.00	0.00	0.00	0.00
\$FIND Call Rate	(Seq)	0.00	0.00	0.00	0.00
	(Key)	0.00	0.00	0.00	0.00
	(RFA)	0.00	0.00	0.00	0.00
\$PUT Call Rate	(Seq)	0.00	0.00	0.00	0.00
	(Key)	0.00	0.00	0.00	0.00
\$READ Call Rate		0.00	0.00	0.00	0.00
\$WRITE Call Rate		0.00	0.00	0.00	0.00
\$UPDATE Call Rate		0.00	0.00	0.00	0.00
\$DELETE Call Rate		0.00	0.00	0.00	0.00
\$TRUNCATE Call Rate		0.00	0.00	0.00	0.00
\$EXTEND Call Rate		0.00	0.00	0.00	0.00
\$FLUSH Call Rate		0.00	0.00	0.00	0.00

OpenVMS Monitor Utility

RMS DATA RATE STATISTICS

on node ELLIS

4-APR-1999 02:27:32.16

(Index) _ELLIS\$DKA300:[ELLIS.RMS]INDEXED.DAT;1

Active Streams:	1	CUR	AVE	MIN	MAX
Total \$GET Call Rate		75.33	75.83	75.33	76.33
Bytes Per \$GET		120.00	120.00	120.00	120.00
Total \$PUT Call Rate		0.00	0.00	0.00	0.00
Bytes Per \$PUT		0.00	0.00	0.00	0.00
\$UPDATE Call Rate		0.00	0.00	0.00	0.00
Bytes Per \$UPDATE		0.00	0.00	0.00	0.00
\$READ Call Rate		0.00	0.00	0.00	0.00
Bytes Per \$READ		0.00	0.00	0.00	0.00
\$WRITE Call Rate		0.00	0.00	0.00	0.00
Bytes Per \$WRITE		0.00	0.00	0.00	0.00
\$TRUNCATE Call Rate		0.00	0.00	0.00	0.00
Blocks Per \$TRUNCATE		0.00	0.00	0.00	0.00
\$EXTEND Call Rate		0.00	0.00	0.00	0.00
Blocks Per \$EXTEND		0.00	0.00	0.00	0.00

OpenVMS Monitor Utility
 RMS FILE LOCKING STATISTICS
 on node ELLIS
 4-APR-1999 02:27:35.16

(Index) _ELLIS\$DKA300:[ELLIS.RMS]INDEXED.DAT;1

Active Streams: 1	CUR	AVE	MIN	MAX
New ENQ Rate	264.33	290.55	264.33	306.00
DEQ Rate	200.33	221.33	200.33	232.33
Converted ENQ Rate	63.66	69.00	63.66	73.33
Blocking AST Rate	0.00	0.00	0.00	0.00
Bucket Split Rate	0.00	0.00	0.00	0.00
Multi Bucket Split Rate	0.00	0.00	0.00	0.00

OpenVMS Monitor Utility
 RMS CACHE STATISTICS
 on node ELLIS
 4-APR-1999 02:27:38.16

(Index) _ELLIS\$DKA300:[ELLIS.RMS]INDEXED.DAT;1

Active Streams: 1	CUR	AVE	MIN	MAX
Local Cache Hit Percent	0.00	0.00	0.00	0.00
Local Cache Attempt Rate	231.00	220.89	198.00	231.00
Global Cache Hit Percent	67.00	68.22	67.00	69.00
Global Cache Attempt Rate	231.00	220.89	198.00	231.00
Global Buf Read I/O Rate	73.66	70.19	64.00	73.66
Global Buf Write I/O Rate	0.00	0.00	0.00	0.00
Local Buf Read I/O Rate	0.00	0.00	0.00	0.00
Local Buf Write I/O Rate	0.00	0.00	0.00	0.00

Understanding CPU Performance

Module 21

Module Outline

Objectives

This module describes issues involved in comparing CPU performance.

Topics

- CISC vs. RISC Architecture
- Understanding CPU Performance
- Data Alignment Issues
- Debugger Issues
- Using the Break/Unaligned Debugger Option
- CPU Performance and Symmetric Multiprocessing
- Galaxy Performance Options
- Galaxy Computing Model
- Galaxy Shared Everything Model
- CPU Migration Under Galaxy
- CPU Controls Using DCL

CISC vs. RISC Architectures

Computer Architectures

Computer architectures describe aspects of a computer system that provide consistency across families of computer products. The fundamental aspect of a computer system is its machine instruction set definition and characteristics. Other aspects of a computer system also define its architecture, such as the way it manages memory and the implementation of its I/O subsystem. The VAX used an architecture referred to as a *Complex Instruction Set Computer* (CISC). Newer architectures, such as Alpha, support a *Reduced Instruction Set Computer* (RISC).

CISC Architecture Goals

- Ease of compiler development
- Large number of instructions for compiler writers
- Smaller programs

CISC Characteristics

- Many processor cycles per instruction, that is, slower instructions due to complexity
- Some noninterruptible instructions
- Pipelining instructions is more difficult

RISC Architecture Goals

- Performance through pipelining, parallelism, and shorter cycle times
- Flexibility and platform independence
- Faster processor development time

RISC Characteristics

- Speed through pipelining, caching, and parallelism
- Large data paths
- Alignment issues
- Lack of microcode
- Typically fewer instructions
- Dependence on good compilers
- Large number of registers

This table shows the CISC vs. RISC type instructions load, operate, and store sequence.

VAX		Alpha	
INCL	X	LDQ	R28, 40(R27)
		LDL	R25, (R28)
		ADDL	R25, 1, R25
		STL	R25, (R28)

Understanding CPU Performance

- Although industry-standard ratings of CPU speeds are made in millions of instructions per second (mips) or mega floating point operations per second (Mflops), these ratings do not truly measure the speed of an individual application.
 - These measures may not be valid for many reasons.
 - The most obvious reason is that the applications may not be CPU bound; they may be I/O bound.
- Another reason these measures may not be valid is that applications designed to take advantage of a system's architecture may run faster than those that ignore architectural considerations.
 - One method of taking advantage of the VAX architecture is to realize the power of its instruction set.
 - The VAX has a move character with three operands (MOVC3) instruction that allows you to move up to 64KB of data in a single instruction; instead of making repeated move-and-loop combinations.
 - Some compilers, such as the FORTRAN compiler, will use loops that move large amounts of data and take advantage of this instruction.
- Another issue in rating CPUs is whether data access is aligned.
 - Many VAX systems fetch on quadword boundaries.
 - Should a data item span quadword boundaries, it may take twice as long to fetch it.
 - Should both source and destination operands overlap quadword boundaries, the fetch cycle may take four times as long.
 - Most compilers will optimize this access on a VAX systems.
 - However, the compilers may be intentionally or unintentionally fooled into a forced alignment that is not optimal.
 - This would happen, for example, when a FORTRAN common area is used.
 - The FORTRAN compiler will not rearrange data stored in common areas, and order of data declaration in these common areas is important.

- The next example illustrates the costs of improper data alignment.
 - The program **NOLONG.FOR** has 5 bytes followed by the longword X.
 - It then initializes X 100,000,000 times to get enough samples to produce numbers that we can work with.
 - The sample run takes roughly 1:30.00 of elapsed CPU time to run (1:31.75 minus 1.32).
 - The **LONG.FOR** program takes roughly 0:45.00 elapsed CPU time to run (2:17.17 minus 1:31.76).
 - The only difference between the two programs is that the data being accessed by **LONG.FOR** is now aligned on a quadword boundary.
- The program runs twice as fast when the data is aligned.
 - Note that the FORTRAN optimizer typically takes care of this performance problem.

VAX Alignment Considerations Example

```
$ type nolong.for
    integer*4      x
    byte    a,b,c,d,e
    common  a,b,c,d,e,x
    do      i=1,100000000
        x=1
    enddo
end

$
$ type long.for
    integer*4      x
    byte    a,b,c,d,e
    common  x,a,b,c,d,e
    do      i=1,100000000
        x=1
    enddo
end

$
$ sh status
Status on  26-NOV-1991 22:59:37.64      Elapsed CPU :   0 00:00:01.32
Buff. I/O :      121    Cur. ws. :   4096    Open files :       0
Dir. I/O :       39    Phys. Mem. :   443    Page Faults :     607

$ r nolong
$ sh status
Status on  24-AUG-1994 23:01:15.08      Elapsed CPU :   0 00:01:31.75
Buff. I/O :      127    Cur. ws. :   4096    Open files :       0
Dir. I/O :       40    Phys. Mem. :   446    Page Faults :     613

$
$ sh status
Status on  26-NOV-1991 23:01:15.13      Elapsed CPU :   0 00:01:31.76
Buff. I/O :      130    Cur. ws. :   4096    Open files :       0
Dir. I/O :       40    Phys. Mem. :   446    Page Faults :     613

$ r long
$ sh status
Status on  24-AUG-1994 23:02:02.22      Elapsed CPU :   0 00:02:17.17
Buff. I/O :      136    Cur. ws. :   4096    Open files :       0
Dir. I/O :       42    Phys. Mem. :   446    Page Faults :     617

$
```

Data Alignment Issues

- Alpha systems load from memory only with longword and quadword data types. The data path on an Alpha system is 64 bits wide and memory fetches are performed on quadword aligned boundaries. Accesses to memory that are not aligned on natural boundaries, such as longword aligned for longword fetches and quadword aligned for quadword fetches, will impose significant performance penalties. True accesses to unaligned data will cause an exception to be generated.
- The following illustrates data accesses that are and are not naturally aligned:

63	56	55	48	47	40	39	32	31	24	23	16	15	8	7	0
XXXX	XXXX	XXXX	XXXX												
	XXXX	XXXX	XXXX	XXXX											

63	56	55	48	47	40	39	32	31	24	23	16	15	8	7	0
YYYY	YYYY	YYYY	YYYY	YYYY	YYYY	YYYY	YYYY	YYYY	YYYY	YYYY	YYYY	YYYY	YYYY	YYYY	YYYY
YYYY	YYYY	YYYY	YYYY	YYYY	YYYY	YYYY	YYYY	YYYY	YYYY	YYYY					
											YYYY	YYYY	YYYY	YYYY	YYYY

- From a performance perspective, on Alpha systems:
 - Bigger data types (ints and longs) are better!
 - Unaligned accesses are very costly if not caught by the compiler.
 - Abuses of pointer casting may limit the compiler's ability to optimize.
- The VAX and DEC C compilers will align nonstructure member data on natural boundaries as part of its optimization process.
- The DEC C compiler will naturally align structure members on natural boundaries; the VAX C compiler will not.
- If you need to access data members that are not naturally aligned in structures, due to file formats, you can override this DEC C optimization by using the **/NOMEMBER_ALIGNMENT** qualifier. However, you will pay a performance penalty for doing so.

- This may also be an issue when using VEST translated images.



Note

Compilers typically generate multiple load instructions and use extract instructions to access the data. An unaligned load attempt will cause an exception, forcing the operating system to correct the fetch.

- Unaligned data accesses cause alignment traps to occur (**SS\$_ALIGN**) because the Alpha systems cannot handle odd byte or word aligned memory accesses. The alignment traps are automatically fixed-up by OpenVMS; however, are very costly in CPU time.

Nonaligned Data Example

```
$
$ type align.c
#include <stdlib.h>
struct    junk
{
    char   a;
    int    y;
    int    x;

};
main()
{
    void    sub(struct junk);

    struct  junk z ={0,0,0};
    int     i;

    for(i=0;i<100000000;i++)
    {
        z.x = 2*i - (i/2) + z.x;
        z.y +=1;
        z.a +=1;
        sub(z);
    }
    return(EXIT_SUCCESS);
}
```

Nonaligned and Compiler Aligned Data Sample Runs

```
$ type sub_align.c
struct      junk
{
    char    a;
    int     y;
    int     x;
};
int sub(struct junk temp)
{
    return(temp.x);
}
$ cc/nomember align
$ cc/nomember sub_align
$ link align,sub_align
$
$ sh status
Status on 24-AUG-1994 03:02:20.06      Elapsed CPU : 0 00:00:02.51
Buff. I/O :      256    Cur. ws. : 2000    Open files :      0
Dir. I/O :      50    Phys. Mem. : 928    Page Faults :    1707
$ r align
$ sh status
Status on 24-AUG-1994 03:03:23.11      Elapsed CPU : 0 00:01:02.35
Buff. I/O :      269    Cur. ws. : 2000    Open files :      0
Dir. I/O :      50    Phys. Mem. : 960    Page Faults :    1741
$
$
$
$
$ cc align
$ cc sub_align
$ link align,sub_align
$
$ sh status
Status on 24-AUG-1994 03:04:45.38      Elapsed CPU : 0 00:01:03.48
Buff. I/O :      371    Cur. ws. : 2000    Open files :      0
Dir. I/O :      77    Phys. Mem. : 832    Page Faults :    2479
$ r align
$ sh status
Status on 24-AUG-1994 03:05:17.54      Elapsed CPU : 0 00:01:32.63
Buff. I/O :      384    Cur. ws. : 2000    Open files :      0
Dir. I/O :      77    Phys. Mem. : 928    Page Faults :    2517
$
```

Aligned Data Example

```

$ type align2.c
#include <stdlib.h>
struct junk
{
    int    y;
    int    x;
    char   a;
};
main()
{
    void    sub(struct junk);
    struct  junk z ={0,0,0};
    int     i;

    for(i=0;i<100000000;i++)
    {
        z.x = 2*i - (i/2) + z.x;
        z.y +=1;
        z.a +=1;
        sub(z);
    }
    return(EXIT_SUCCESS);
}
$ type sub_align2.c
struct junk
{
    int    y;
    int    x;
    char   a;
};
int    sub(struct junk temp)
{
    return(temp.x);
}
$ cc/nomember align2
$ cc/nomember sub_align2
$ link align2,sub_align2
$ sh status
Status on 24-AUG-1994 03:10:14.04      Elapsed CPU :   0 00:01:34.83
Buff. I/O :      637    Cur. ws. :   2000    Open files :      0
Dir. I/O :      122    Phys. Mem. :   896    Page Faults :   3801
$ r align2
$ sh status
Status on 24-AUG-1994 03:10:47.72      Elapsed CPU :   0 00:02:03.99
Buff. I/O :      650    Cur. ws. :   2000    Open files :      0
Dir. I/O :      122    Phys. Mem. :   928    Page Faults :   3838
$

```

Badly Aligned Macro Example

- Code for align, compiled with nomember alignment:

```

1   1525      for(i=0;i<100000000;i++)
      2   1526      {
      2   1527          z.x = 2*i - (i/2) + z.x;
      2   1528          z.y +=1;
      2   1529          z.a +=1;

```

- Concentrate on code used to implement line 1528:

```

00E0      L$2:
2E1D000D      00E0      LDQ_U   R16, 13(FP)
                        ; 001527

2E3D0010      00E4      LDQ_U   R17, 16(FP)
203D000D      00E8      LDA     R1, 13(FP)

```

- Load quadword unaligned 12 off PV to R18:

```

2E5D000C      00EC      LDQ_U   R18, 12(FP)
                        ; 001528

-----
4A0104D0      00F0      EXTLL   R16, R1, R16
                        ; 001527
406019A0      00F4      CMPLT   i, 0, R0
; R3, 0, R0
4A210D51      00F8      EXTLH   R17, R1, R17
40600000      00FC      ADDL    i, R0, R0
; R3, R0, R0
48003780      0100      SRA     R0, 1, R0
46300401      0104      BIS     R17, R16, R1

```

- Get start of longword into R17. Get start of longword into R16. Get high byte into R20:

```

2E3D0009      0108      LDQ_U   R17, 9(FP)
                        ; 001528
221D0009      010C      LDA     R16, 9(FP)
4A500D54      0110      EXTLH   R18, R16, R20

-----
40800120      0114      SUBL    R4, R0, R0

```

- Get low 3 bytes into R19:

```

; 001527

4A3004D3      0118      EXTLL   R17, R16, R19
                        ; 001528

-----
40010000      011C      ADDL    R0, R1, R0
                        ; 001527

```

- Mask low byte into R18. Set high three bytes into R19. Set up for write to memory, with original data. Finally do add. Set up R20 for write:

```

4A500C52      0120      MSKLH   R18, R16, R18
                                   ; 001528
46930413      0124      BIS      R20, R19, R19
4A300451      0128      MSKLL   R17, R16, R17
42603013      012C      ADDL     R19, 1, R19
4A700CF4      0130      INSLH   R19, R16, R20
-----
22BD000D      0134      LDA      R21, 13(FP)
                                   ; 001527

```

- Set up R19 for write to memory. Set up original data with updated data. Write it to memory with two store quads:

```

4A700573      0138      INSL    R19, R16, R19
                                   ; 001528
46540412      013C      BIS      R18, R20, R18
46330411      0140      BIS      R17, R19, R17
3E5D000C      0144      STQ_U    R18, 12(FP)
3E3D0009      0148      STQ_U    R17, 9(FP)
-----

```

...

Well Aligned Macro Example

- Code for align2 with good alignment:

```

1   1525           for(i=0;i<100000000;i++)
2   1526           {
2   1527               z.x = 2*i - (i/2) + z.x;
2   1528               z.y +=1;
2   1529               z.a +=1;

```

- Again, concentrate on code for line 1528:

```

00E0          L$2:
A01D0008      00E0          LDL      R0, z
; R0, 8(FP)          ; 001529
406019A1      00E4          CMPLT   i, 0, R1
; R3, 0, R1          ; 001527
40610001      00E8          ADDL    i, R1, R1
; R3, R1, R1

```

- Get 12 off PV into R17:

```

A23D000C      00EC          LDL      R17, 12(FP)
; 001528
-----
A29D0010      00F0          LDL      R20, 16(FP)
; 001527
48203781      00F4          SRA      R1, 1, R1
40003410      00F8          ADDQ     R0, 1, R16
; 001529
441FF112      00FC          BIC      R0, 255, R18
461FF013      0100          AND      R16, 255, R19
40810121      0104          SUBL     R4, R1, R1
; 001527
46530400      0108          BIS      R18, R19, R0
; 001529
A7420020      010C          LDQ      R26, 32(R2)
; 001530

```

- Do add of 1:

```

42203011      0110          ADDL     R17, 1, R17
; 001528
-----
40340001      0114          ADDL     R1, R20, R1
; 001527
B01D0008      0118          STL      R0, z
; R0, 8(FP)          ; 001529
40805004      011C          ADDL     R4, 2, R4
; 001525

```

■ Store it back in memory:

B23D000C	0120	STL	R17, 12(FP) ; 001528

47E05419	0124	MOV	2, R25 ; 001530
B03D0010	0128	STL	R1, 16(FP) ; 001527 ; 001527
40603003	012C	ADDL	i, 1, i ; 001525
; R3, 1, R3			
A63D0010	0130	LDQ	R17, 16(FP) ; 001530
A61D0008	0134	LDQ	R16, z
; R16, 8(FP)			
A7620028	0138	LDQ	R27, 40(R2)

C Code with Bad Alignment

```
$
$ type force_bad_align.c
#include <stdio.h>
#include <stdlib.h>

void stub(long *);
int main(void)
{
    short  stuff[100];
    long   *ptr;
    int    i, j;

    for(j=0;j<1000000;j++)
    {
        ptr = (long *)&stuff[1];
        *ptr = 0;
        for(i = 0; i < 40; i++)
        {
            *ptr++ = *ptr * i;
        }
        stub(ptr);
    }
    return(EXIT_SUCCESS);
}
$
$ type stub.c
void stub(long *x)
{
}
$ cc force_bad_align
$ cc stub
$ link force_bad_align,stub
$
$ show status
Status on 31-AUG-1994 22:34:17.45      Elapsed CPU : 0 00:00:01.72
Buff. I/O :      193    Cur. ws. :    2000    Open files :      0
Dir. I/O :      103    Phys. Mem. :    960    Page Faults :    969
$ r force_bad_align
$ sh status
Status on 31-AUG-1994 22:35:39.51      Elapsed CPU : 0 00:01:13.78
Buff. I/O :      206    Cur. ws. :    2000    Open files :      0
Dir. I/O :      103    Phys. Mem. :    912    Page Faults :   1035
$
```


C Small Data Accesses Issues

```

$ type shorts.c
#include <stdio.h>
#include <stdlib.h>

void stub(long *);
int main(void)
{
    short stuff[100];
    short *ptr;
    int i, j;

    for(j=0;j<1000000;j++)
    {
        ptr = &stuff[1];
        *ptr = 0;
        for(i = 0; i < 80; i++)
        {
            *ptr++ = *ptr * i;
        }
        stub((long *)ptr);
    }
    return(EXIT_SUCCESS);
}
$ cc shorts
$ link shorts,stub
$ sh status
Status on 31-AUG-1994 22:35:44.16 Elapsed CPU : 0 00:01:15.01
Buff. I/O : 321 Cur. ws. : 2000 Open files : 0
Dir. I/O : 126 Phys. Mem. : 816 Page Faults : 1576
$ r shorts
$ sh status
Status on 31-AUG-1994 22:36:15.84 Elapsed CPU : 0 00:01:39.97
Buff. I/O : 334 Cur. ws. : 2000 Open files : 0
Dir. I/O : 126 Phys. Mem. : 912 Page Faults : 1615
$

```

C Code with Good Alignment

```
$
$ type good_align.c
#include <stdio.h>
#include <stdlib.h>

void stub(long *);
int main(void)
{
    short stuff[100];
    long *ptr;
    int i, j;

    for(j=0;j<1000000;j++)
    {
        ptr = (long *)stuff;
        *ptr = 0;
        for(i = 0; i < 40; i++)
        {
            *ptr++ = *ptr * i;
        }
        stub(ptr);
    }
    return(EXIT_SUCCESS);
}
$ cc good_align
$ link good_align,stub
$
$ sh status
Status on 31-AUG-1994 22:36:39.86 Elapsed CPU : 0 00:01:41.22
Buff. I/O : 451 Cur. ws. : 2000 Open files : 0
Dir. I/O : 152 Phys. Mem. : 816 Page Faults : 2143
$ r good_align
$ sh status
Status on 31-AUG-1994 22:36:51.50 Elapsed CPU : 0 00:01:48.11
Buff. I/O : 464 Cur. ws. : 2000 Open files : 0
Dir. I/O : 152 Phys. Mem. : 912 Page Faults : 2182
$
$
```

Debugger Issues

- The debugger allows you to break on alignment traps, using:
`SET BREAK/UNALIGNED`
- This mechanism can be used on particularly slow applications to determine whether alignment traps are slowing down your application.
- The debugger might move back and forth through instructions when single stepping through instructions. The **/NOOPTIMIZE** option can resolve some of this shuffling. Note that the shuffling is generally in a forward fashion.

Using the Break/Unaligned Debugger Option

```
676      long   *ptr;
677      int     i, j;
678
679      for(j=0;j<1000000;j++)
680      {
681          ptr = (long *)&stuff[1];
682          *ptr = 0;
683          for(i = 0; i < 40; i++)
684          {
> 685              *ptr++ = *ptr * i;
686          }
687          stub(ptr);
688      }
689      return(EXIT_SUCCESS);
690  }
```

DBG> **set break/unaligned**

DBG> **go**

break at routine FORCE_BAD_ALIGN\main

DBG> **go**

Unaligned data access: virtual address = 7F9719AA, PC = 000200BC

break on unaligned data trap preceding FORCE_BAD_ALIGN\main\%LINE 681

DBG>

DBG> **go**

Unaligned data access: virtual address = 7F9719B6, PC = 000200E8

break on unaligned data trap preceding FORCE_BAD_ALIGN\main\%LINE 685+16

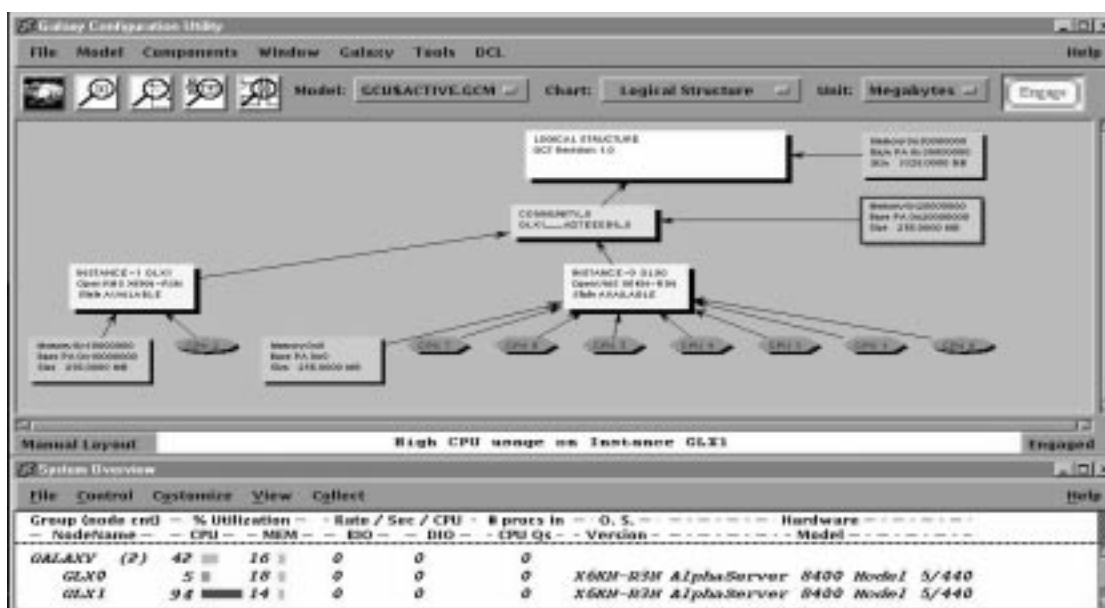
DBG>

DBG>

CPU Performance and Symmetric Multiprocessing

- One thing to bear in mind when moving to a symmetric multiprocessing (SMP) system is that an OpenVMS scheduling process runs on one CPU at a time.
 - This means that when you move to an N-speed SMP system with four CPUs, you can expect a single CPU-bound application to run N-divided-by-4 times faster, not N times faster.
 - The speed ratings for these systems are based on the assumption that they are running in a multiuser configuration.
 - Using kernel/pthreads can speed up an individual application.
- Another thing to bear in mind with SMP systems is that CPU time spent processing device interrupts is not currently distributed across CPUs.
 - A single processor, a somewhat loosely defined "primary" processor, processes all interrupts.
 - The architecture does not dictate that this be the case, but practical performance studies have proved that this is the most effective method of processing interrupts at this time.
- One potential problem with the lack of distribution of interrupt stack time is that a system processing interrupts can block another CPU from accessing a synchronization entity called a *spinlock*.
 - This can be tracked by monitoring the multiprocessing (MP) synchronization time.
 - This topic will be more formally addressed in the discussion of time spent in processor modes.
- One level of Interrupt state distribution that can be performed is through "Fast Path" CI support, which will dedicate CI Interrupt state time to a CPU other than the Primary. Fast Path is enabled by default with the **SYSGEN** parameters **FAST_PATH** (boolean) and **IO_PREFER_CPUS** (bitmask).

Galaxy Performance Options



OpenVMS Galaxy systems provide a new multiprocessing architecture designed to address current computing environments and to evolve into the next generation of hardware architecture.

The OpenVMS Galaxy computing model is defined as *Adaptive Partitioned MultiProcessing* (APMP).

Galaxy Computing Model

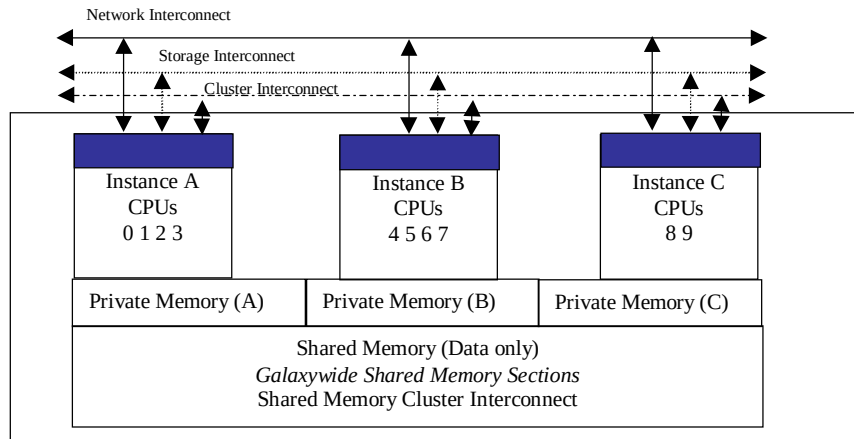
Partitioned

- Groups of CPUs, along with memory and I/O resources form *Instances* of the OpenVMS operating system, which execute cooperatively, but distinctly, within the same computer.
- An Instance of OpenVMS is not unlike an SMP OpenVMS Cluster *Node* in that it is a separate copy of OpenVMS that can share file system related information through Distributed Lock Management.
- It is unlike a OpenVMS Cluster node in the following ways:
 - Availability (all instances within a computer share the same power supply)
 - It **can** share memory (and possibly other resources)

Adaptive

Resources, such as CPUs, can be reconfigured dynamically between Instances to meet business computing needs.

Galaxy Shared Everything Model



Shared Everything Computing Model

Benefits:

- Smaller footprint
- Multiple (upgrade) versions of OpenVMS in same Box
- Adaptive - CPUs may be reassigned and files/records are shared
- Performance may improve for all applications through dynamic reconfiguration of CPUs in a homogenous computing and file system environment
- Some Availability (Rolling Upgrade/Instances crash independently)
- Lesser impact of SMP overhead, since isolated to instance
- Possibly lesser impact of cluster communications through shared memory cluster interconnect
- Easier to administer with a single user profile environment and a single shared system disk
- Single security domain

CPU Migration Under Galaxy

- CPUs can be migrated from one instance to another using:


```
$ STOP/CPU[/ALL]/MIGRATE=[SCSNODE/PARTITION-ID] [cpu-id]
```
- The following CPUs cannot be stopped and migrated:
 - The Primary CPU
 - A CPU with "Fast-path" enabled
 - A CPU for which a process has affinity
- CPUs can be migrated using the Galaxy Configuration Utility (GCU), using a "drag and drop" GUI interface.
- Sample CPU migration using DCL:

```
jygal> show cpu
```

```
JYGAL, a AlphaServer 4100 5/300 2MB
Multiprocessing is ENABLED. Streamlined synchronization image loaded.
Minimum multiprocessing revision levels: CPU = 1
```

```
PRIMARY CPU = 00
```

```
CPU sets:
```

```
Active      00 02 03
Configure   00 02 03
Potential   00 01 02 03
Autostart    00 01 02 03
Failover     None
```

```
jygal> stop/cpu/migrate=jygal2 2,3
```

```
%SYSTEM-I-CPUSTOPPING, trying to stop CPU 2 after it reaches quiescent state
%SYSTEM-I-CPUSTOPPING, trying to stop CPU 3 after it reaches quiescent state
```

Sample CPU Migration Using DCL

- View from instance JYGAL.

```
jygal> show cpu
```

```
JYGAL, a AlphaServer 4100 5/300 2MB
Multiprocessing is ENABLED. Streamlined synchronization image loaded.
Minimum multiprocessing revision levels: CPU = 1
```

```
PRIMARY CPU = 00
```

```
CPU sets:
```

Active	00
Configure	00
Potential	00 01 02 03
Autostart	00 01 02 03
Failover	None

```
jygal>
```

- View from instance JYGAL2.

```
jygal2> show cpu
```

```
JYGAL2, a AlphaServer 4100 5/300 2MB
Multiprocessing is ENABLED. Streamlined synchronization image loaded.
Minimum multiprocessing revision levels: CPU = 1
```

```
PRIMARY CPU = 01
```

```
CPU sets:
```

Active	01 02 03
Configure	01 02 03
Potential	00 01 02 03
Autostart	00 01 02 03
Failover	None

```
jygal2> stop/cpu/migrate=jygal 2,3
```

```
%SYSTEM-I-CPUSTOPPING, trying to stop CPU 2 after it reaches quiescent state
```

```
%SYSTEM-I-CPUSTOPPING, trying to stop CPU 3 after it reaches quiescent state
```

```
jygal2>
```

CPU Controls Using DCL

- A CPU can be enabled or disabled for auto-start, by migration or failover using:
`$ SET CPU/[NO]AUTOSTART cpu-id[,cpu-id...]`
- A CPU can be designated as a failover CPU, such that it will automatically be added to another instance when the current bugchecks, before crash is completely written. The failover will not occur until the crash dump information has been written. To designate a CPU as a failover CPU, use:
`$ SET CPU/FAILOVER=instance cpu-id[,cpu-id...]`

Understanding Time Spent in Modes

Module 22

Module Outline

Objectives

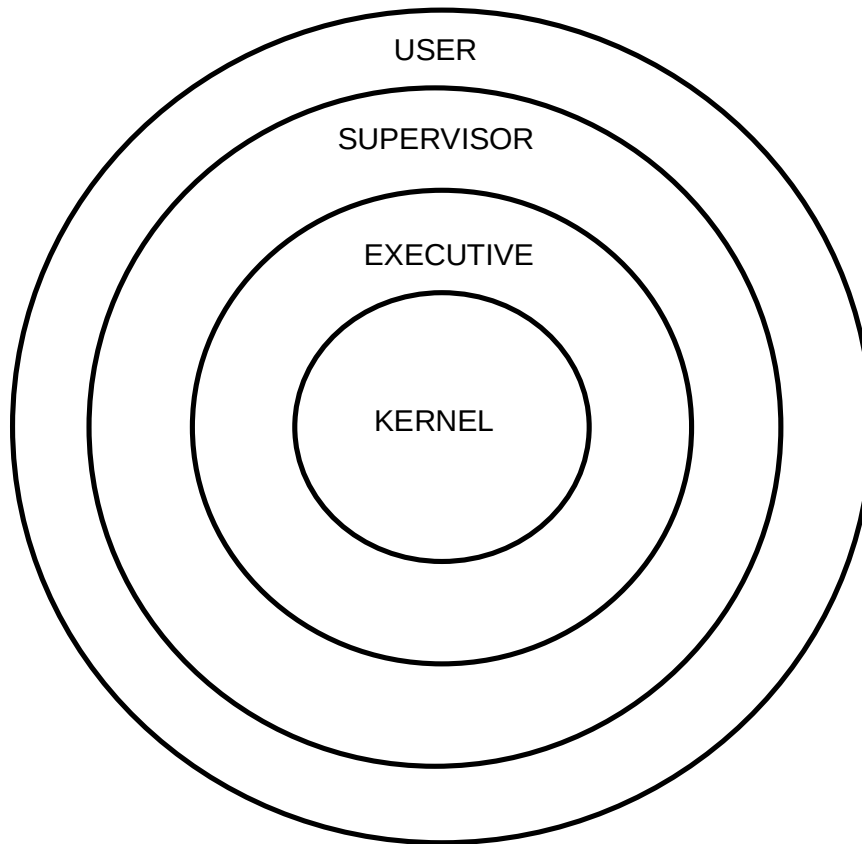
This module describes activities that occur in each of the given processor modes. It will allow the student to determine how CPU time is being spent and places to look at for overhead reduction.

Topics

- OpenVMS Access Modes
 - Kernel Mode
 - Executive Mode
 - Supervisor Mode
 - User Mode
- Interrupt Stack/State Time
- MP Synchronization Time
- Idle Time
- Compatibility Mode Time
- Examining Time Spent in Modes
- Analyzing Time Spent in Modes

OpenVMS Access Modes

- When your system has become CPU bound, you may want to investigate where the CPU time is being spent.
 - If the CPU time that you are using is due to system overhead, you may be able to alleviate some of this overhead time and free up the CPU for more productive uses.
- To aid you in determining where CPU time is being spent, the **\$MONITOR MODES** tells you the number of 10-ms "ticks" being charged in the various processor modes.
 - Understanding what is happening in each of these modes can help you to determine how the CPU time is being spent and possibly allow you to change characteristics of the system to free up this time.
- OpenVMS supports four processor modes for execution: *kernel*, *executive*, *supervisor*, and *user*.
 - The primary purpose of these modes is to protect the system from the user and the users from themselves.
 - Certain activities tend to operate in these given modes. Time spent in each mode operates in process context and therefore is being charged against some process.



- **Kernel Mode**—System Services, Pager, Scheduler, device drivers, and so on.
- **Executive Mode**—Record Management Services (RMS)
- **Supervisor Mode**—Command Language Interpreter (DCL)
- **User Mode**—User programs, utilities, commands (not handled internally by DCL)

Kernel Mode

- Kernel mode time is generally set aside for operating system support.
 - The operating system support can be requested explicitly by the process, by its issuance of a system service call such as **SYSS\$QIO**, **SYSS\$CREPRC**, or **SYSS\$ENQ**.
 - The overhead in this case would be dictated by the design of the application and would require design modifications to alleviate it.
- Kernel mode time can also occur implicitly, because of system support requirements.
- Implicit kernel mode time activities include the following:

Page Faults

- A soft page fault is resolved in kernel mode.
- If you are incurring a high free or modified page list fault rate, you may want to investigate your working set sizes.
- If the working set list sizes are artificially small, you may be able to decrease these faults by increasing the working set list sizes of processes.

Lock Requests

- When accessing a shared file, the process takes out lock requests to synchronize access to these files.
- Locks are required for proper synchronization to shared files.
- In fact, when you are buffering files effectively, the locking rates may go up, because you are not waiting for I/O to complete before processing a given record.
- If you are accessing files for read-only access, make sure that the files have not been open for shared read/write access, because the process will perform unnecessary locking.
- In general, all you can do from a system perspective is write off this overhead.

Logical Name Translations

- These translations are also required.
- Do not expect to eliminate them, because they give you device and file independence.
- Bear in mind, however, that each file open takes two or more logical name translations.
- Minimizing file opens by proper application design will decrease the overhead of logical name translations.

I/O Activity

- The first portion of an I/O request is processed by the **SYSSQIO** system service.
- You can minimize some of this overhead by performing less-frequent, larger I/O operations.

File System Requests

- The file system code for the extended **QIO** procedures (**XQP**) operates in kernel mode.
- This time may be tracked in the **\$MONITOR FCP** display under the field File System Tics.
- Each tic corresponds to 10 ms of CPU time being charged for processing file system requests.
- If this time is becoming excessive, you can look for poorly designed applications or window turns, caused by accessing fragmented files.
- Bear in mind that file system tics include time associated with I/O and lock requests.

Executive Mode

Executive mode activities primarily include Record Management Services (RMS) processing.

- This time indicates that you are most probably accessing files.
- You will not likely be able to eliminate the access of files, so this time is unavoidable.
- In fact, if you are effectively buffering the I/O for file accessing, this time is likely to go up because you no longer have to wait for file I/O to complete before using the CPU.
- Another potential cause of high executive mode time is the overuse of access control list entries.
- These entries are translated using the rights list database in executive mode.

Supervisor Mode

Supervisor mode activities are most commonly related to command language interpreter (CLI) operations.

- DCL is the default, and CLI is the most common cause of these activities.
- If the supervisor mode time is exceeding 10 percent of your CPU time, this is probably caused by overuse of command procedures.
- Although command procedures are easy to write and modify, one of the performance implications of writing them is that each command must be interpreted in supervisor mode before it can be executed.
- Where possible, write programs instead of command procedures to perform commonly executed tasks.

User Mode

This time is made up of your applications' and layered products' processing.

- A high user mode time is an indication that you need to improve your applications' algorithms or buy a faster CPU if you want to free up the CPU.
- If you are noticing response time degradation due to a saturated CPU, you can consider lowering the interval for rescheduling by setting a smaller value for the **QUANTUM SYSGEN** parameter.
- This will force throughput-oriented, CPU-intensive applications to give up the CPU more frequently, making it available for response time-oriented interactive users.
- The cost for this smaller value will be some throughput degradation due to the relatively small overhead of rescheduling more frequently.
- If the CPU time is being used by throughput-intensive applications, consider running them in batch mode at a lower priority, in the range from 1 to 3.
 - This will allow these jobs to get available CPU time but not block the interactive job's need for the CPU.
 - To force an application to be run in batch, you may use the following commands:

```
$SET FILE/ACL=(IDENTIFIER=*,ACCESS=NONE) CRUNCH.EXE
```

```
$SET FILE/ACL=(IDENTIFIER=BATCH,ACCESS=EXECUTE) CRUNCH.EXE
```

- This will force the user who wants to run this application to run it in batch mode or not at all.

Interrupt Stack/State Time

- Interrupt stack time (Interrupt State Time on Alpha) consists of processor time spent processing interrupts.
 - The interrupts are typically initiated by a hardware request.
 - This request may be an I/O completion or a clock interrupt.
 - To schedule lower priority processing within the context of an interrupt, the software processing a hardware interrupt may request a software interrupt.
 - Software interrupts include the lowering of the priority of a device driver processing an interrupt (fork processing), software timer processing, I/O completion, and rescheduling.
- Both hardware- and software-requested interrupts have a special priority associated with them.
 - This priority is referred to as *interrupt priority level* (**IPL**).
 - Activities processed at an elevated **IPL** (that is, an **IPL** greater than 2) block all process-related activities and inhibit scheduling.
- Interrupts are preemptive in blocking all process activities and are, therefore, causes of response time degradation.
 - Of the CPU time spent processing activities in the given modes, interrupt stack time is the most costly because it affects all processes on the system, not just the current process.
- Causes of interrupt stack time are as follows:
 - Processing I/O requests
 - ♦ The most common cause of excessive interrupt stack time is I/O-related activity.
 - ♦ Minimizing the number of I/O requests by requesting larger I/Os less frequently is probably the best solution to this problem.
 - Rescheduling and Timer Processing
 - ♦ The primary reason for this increase in overhead is a lower setting for the **SYSGEN** parameter **QUANTUM**.
 - ♦ **QUANTUM** is measured in units of 10-ms intervals.
 - ♦ Thus, even at its smallest setting of 2, **QUANTUM** should not cost more than 2 to 3 percent additional interrupt stack time.

- Distributed lock requests
 - The processing of distributed locks in a OpenVMS Cluster is performed on the interrupt stack.
 - Distributed locking strategies will be discussed later.
- Mass Storage Control Protocol (MSCP) serving
 - The serving of disk blocks in a OpenVMS Cluster to a remote node is performed on the interrupt stack.
 - Interrupt stack time costs for serving these blocks can be fairly high.
 - With a large number of server requests, a boot member or server node can become quickly bogged down in terms of CPU time.

MP Synchronization Time

- When an SMP system is in operation, several CPUs may be interested in the same information within system executive databases.
 - To guarantee proper synchronization, the systems must acquire a spinlock on the given database before accessing it.
- A spinlock is a fancy data structure that contains one very important bit.
 - We will call this bit the *in-use* bit.
 - Each executive database (memory management, I/O fork processing, scheduling, and so on) has a spinlock assigned to it.
 - Before accessing the given database, the CPU on which the interested code is running, "requests" the spinlock for the database.
 - The request is processed by checking the "in-use" bit.
 - If the bit is clear, the CPU uses a memory interlocked instruction to set the bit.
 - Once the bit is set, the CPU can access the database.
 - When the CPU is done accessing the database, it is responsible for clearing the in-use bit.
- When the in-use bit is set, the CPU loops, or spins, checking for the in-use bit to become clear.
 - Once the bit is clear, the CPU may acquire access to the given database.
 - This time spent waiting for the in-use bit to become clear translates into lost CPU time.
 - This lost CPU time is charged as **MP** synchronization time.
 - On a typical system, this time does not exceed 5 percent of the overall CPU time.
- The CPU time spent waiting may have been requested on the interrupt stack or in kernel mode at an elevated **IPL**.
 - If this time becomes excessive - more than 5 percent - you should investigate minimizing the interrupt stack time by minimizing I/Os, distributed lock requests, and **MSCP** serving.

Idle Time

Idle time is charged in the rescheduling interrupt service routine while the system is waiting for a process to become computable.

- From a throughput standpoint, idle time is to be avoided, because you have lost resource time, that is, CPU time, waiting for some event such as **I/O** to complete.
- From a response time standpoint, some idle time is good because it allows for surplus resource time, again CPU time, to be available when it is needed.
 - This surplus prevents response time degradation.
 - Unless there is a low-priority batch process that is sopping up the extra CPU cycles, attempt to shoot for 25 to 40 percent idle time, depending on the response time demands of the system.

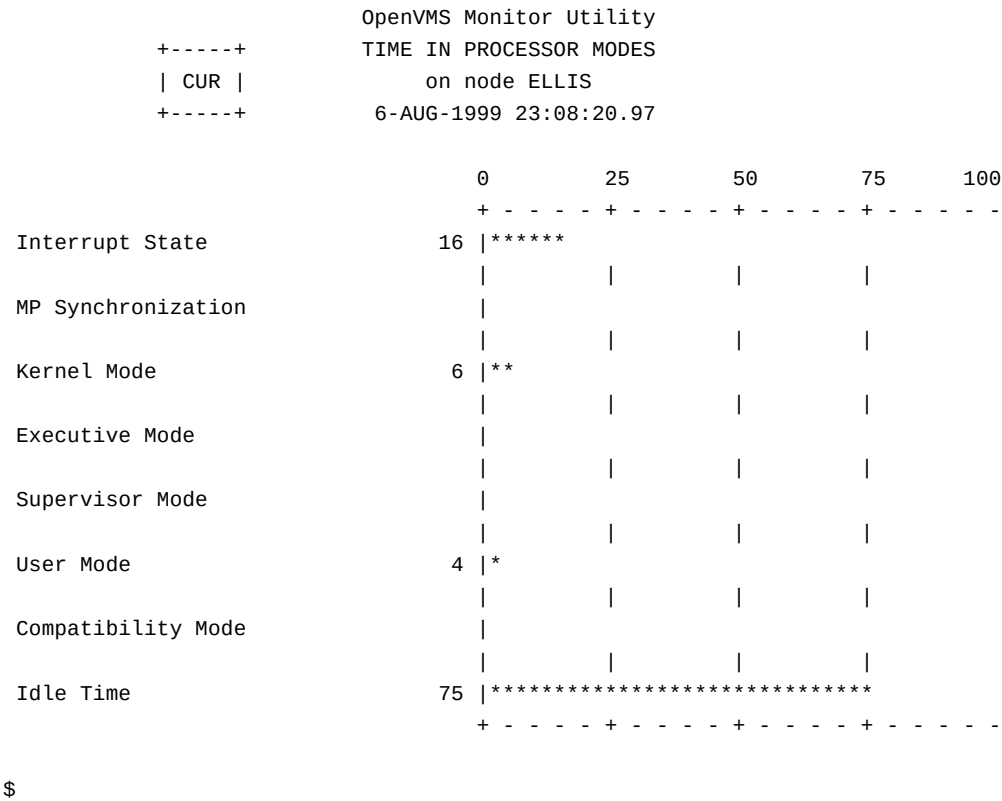
Compatibility Mode Time

Compatibility mode time is spent executing PDP-11 instructions in compatibility mode.

- This time is now rare, because most systems no longer support compatibility mode.
- If you do experience this time, convert your compatibility mode applications to native mode as quickly as possible.

Examining Time Spent in Access Modes

```
$ spawn/nowait r [.perf]thrasher
%DCL-S-SPAWNED, process ELLIS_2 spawned
$ MON MODES
```



Analyzing Time Spent in Modes

How much time should be spent in a given mode?

Traditionally, certain rules of thumb have been given in an attempt to aid you in determining how much time spent in a given mode can be considered too much.

- Although these rules of thumb can be of some assistance to the novice, they provide little aid in solving problems.
- It is much better to attempt to determine why the time is being spent in the given mode and attempt to alleviate this overhead, if possible, than to simply point a finger at the CPU time being spent in that mode as being too much.
- For instance, a maximum of 20 to 25 percent kernel mode time was traditionally considered the comfort zone.
 - However, if you have tuned the RMS activities within your system, you may find that you are spending much less time waiting for I/O to complete.
 - With this wait time restriction lifted, you now can process records more quickly.
 - If you are accessing shared files, the number of locks that you can process will go up as well.
 - This implies that you will be spending more time in kernel mode, perhaps as much as 30 to 35 percent of your CPU time.
 - This means that you are getting more work done, and above and beyond this improvement, the process/user that is processing the records is being charged for the CPU time and is not blocking other users.
 - So in this case, there is little that you can do about the higher kernel mode time, and in fact, it may be better overall for your system to be experiencing this time.
- In similar fashion, a system that is a boot member in a **LAVc** is much more likely to experience additional interrupt stack time than a VAX that is running standalone.
 - You should realize the impact of these times and attack that which is reasonable and which will yield the most desired results.

OpenVMS Scheduling and Priorities

Module 23

Module Outline

Objectives

This module describes the scheduling characteristics of OpenVMS and priorities of processes. Students will understand how to set base priorities and OpenVMS manipulation of these priorities.

Topics

- Quantum and CPU Time
- Priorities
- Waiting States
- States of Concern

Quantum and CPU Time

QUANTUM (which is the rescheduling interval for OpenVMS) and CPU time are charged by the hardware clock interrupt service routine, which runs every ~10 ms.

- If OpenVMS is not processing an interrupt when the clock interrupts, the current process gets charged for 10 ms of CPU time. If a process reaches **QUANTUM** end, it has become a candidate for rescheduling.
- Note that although the setting for **QUANTUM** is made system-wide, each process gets charged independently for **QUANTUM**.
- When a process enters a wait state, it will run later with whatever **QUANTUM** it had remaining.
- The default setting for **QUANTUM** is 20 in 10-ms units.
- This setting translates to a 0.2-second interval of CPU time before the current process must give up the CPU to a process of equal priority.
- When a large number of processes are competing for the CPU, and response time is the primary goal in your tuning strategy, you might consider a lower setting for **QUANTUM**.
 - Realize that **QUANTUM** is set the same for all processors regardless of speed, although the rescheduling overhead becomes smaller and smaller as processors get faster.
 - Bear in mind that on OpenVMS, rescheduling is a CPU-bound activity and does not require I/O as it may on other systems.
 - If **QUANTUM** is set to 20 (or 0.2 seconds), only five CPU-bound processes of the same priority can use the processor per second.
 - If **QUANTUM** has been lowered to 4 (or 0.04 seconds), 25 processes that are CPU bound and are of equal priority can use the CPU per second.
- Do not expect huge gains in performance by lowering the **QUANTUM**, but expect a more responsive system when the CPU is a resource of contention.
 - If you have very few computable processes, the setting for **QUANTUM** will have almost no effect on response time performance.
- There are two primary costs of lowering **QUANTUM**:
 - Rescheduling overhead
 - Throughput degradation

Priorities

OpenVMS schedules processes on the basis of priority.

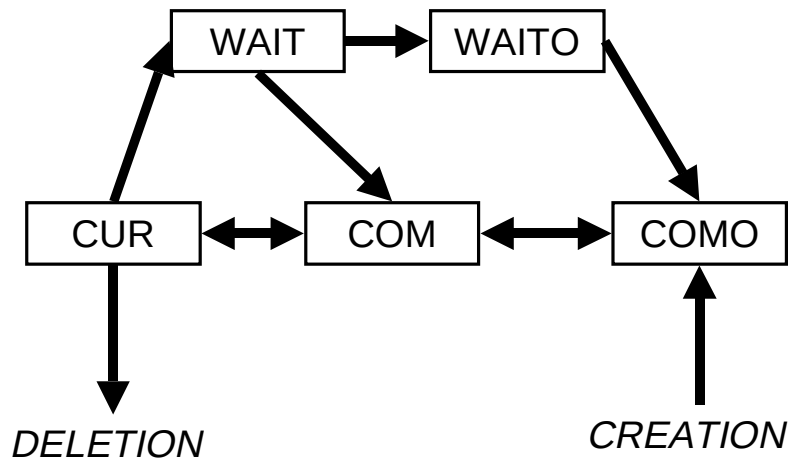
- A process that has had an event complete will get the CPU if it is of sufficiently higher priority than the current process.
- These priorities are assigned at process creation time.
- The assigned priority is referred to as the *base priority* of the process. OpenVMS can adjust the priority on the basis of the activities in which the process has been involved.
- The scheduling priorities range from 0 to 31 (VAX) or 0 to 63 (Alpha).
- They are separated into two ranges: 0 to 15 are referred to as the *timeshare priorities*, and 16 to 31/63 are referred to as the *real-time priorities*.

Differences in Scheduling Between Priority Ranges

Timeshare Priority	Real-Time Priority
Round-robin at equal priorities	No round-robin scheduling
May enter a wait state	May enter a wait state
Preemptive at greater priorities by default (adjustable by the SYSGEN parameter PRIORITY_OFFSET)	Preemptive at greater priorities
Receive priority boosts	No priority boosts
Receive automatic working set adjustment	No automatic working set adjustment

Scheduling States

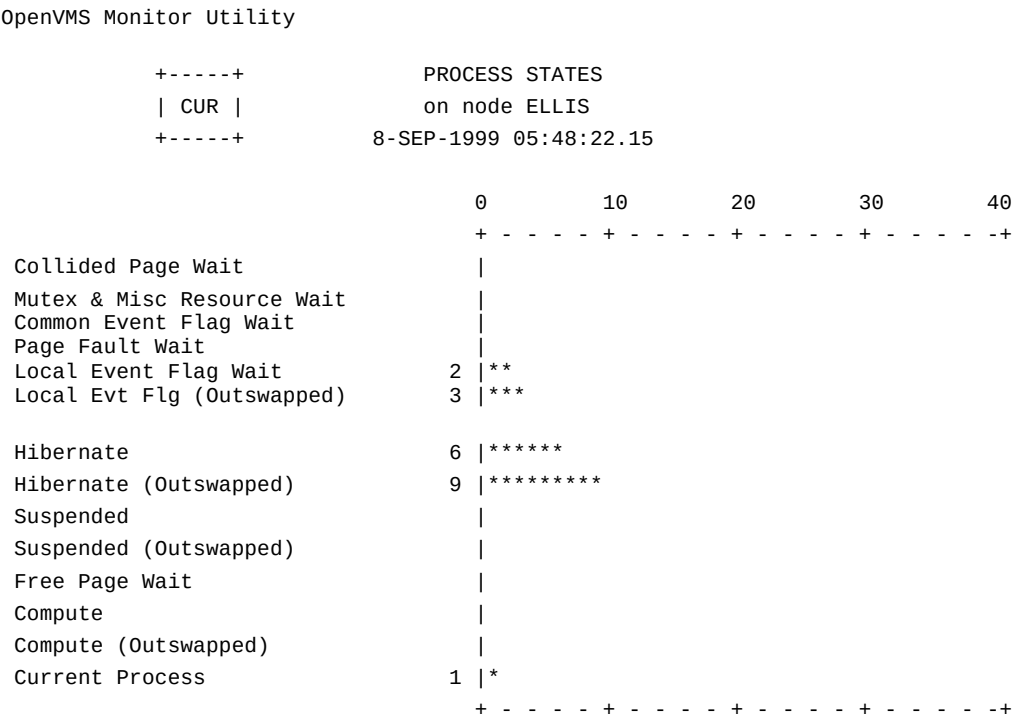
- Like any state-oriented scheduling system, OpenVMS supports three fundamental states: current (**CUR**), computable (**COM**), and waiting (**WAIT**), as illustrated in the next example.
 - There are up to **N** current processes running on the system, where **N** is the number of **SMP** CPUs on the system.
 - If the system is idle, there may be no current processes.
 - The current process is using the CPU and its resources, such as registers.
 - Only one process can use a given CPU at a time.
 - The **N** highest priority process should be using the CPU at any given time.
- General scheduling states



To become current, a process must first be computable.

- Unlike waiting processes, computable processes are tracked by their priority.
- A computable process is ready to run, but is currently blocked from using the CPU by a higher priority process.

MONITOR States



Computable Processes

- The process will receive the CPU and become current when either of the following occurs:
 - It has become the highest priority process.
 - ♦ This can happen when a process of greater priority enters a waiting state. Another process has an event reported on it. It is made computable with a priority greater by 1 or more (if it is a real-time priority process), or when it has a priority greater than that of the current process plus the **SYSGEN** parameter **PRIORITY_OFFSET**.
 - The current process is of equal priority, is running in the timeshare range, and has reached **QUANTUM** end. We reschedule using *round-robin scheduling*.
- The parameter **PRIORITY_OFFSET** was added in version 5.3 of OpenVMS.
 - Setting this parameter higher than the default setting of 0 forces a process that has become computable to wait until **QUANTUM** end to get the CPU rather than preempt the current process, unless it is sufficiently important (that is, its priority is greater than the current processes by **PRIORITY_OFFSET**).
 - This behavior trades off better preservation and utilization of the CPU caches against potential response time degradation where the setting for **QUANTUM** is large.
- Processes become computable when they are preempted by higher priority processes, when they have been round-robin scheduled, when they were previously waiting and an event was reported to signal the end of the wait, and when they are inswapped from a computable outswapped state.

Waiting States

- A large number of computable processes indicates that the CPU has become a bottleneck and a point of contention for response time.
 - If there are only a few computable processes (one to three), the CPU is not a bottleneck, unless you are striving for better throughput for these processes.
- A process enters a waiting state when it must wait for some activity to complete before it can effectively use the CPU.
 - The waiting states are grouped into two categories: *voluntary* and *involuntary*.
 - Voluntary waiting states occur when the process (or some other process) requests to give up the CPU until some event completes.
 - Involuntary waiting states are caused when OpenVMS must wait for a system activity to complete or a system resource to become available before the process can run again.
 - Involuntary waiting states are typically indications of performance or resource problems.
 - The voluntary waiting states are not normally an indication of a performance problem, with the possible exception of the **LEF** state, which is when the process is waiting for a local event flag to be set.

Voluntary Wait States

State	Reason to Enter	Reason to Leave
SUSP	The process was suspended by another process, using the \$SUSPND system service or the \$SET PROCESS/SUSPEND DCL command.	The process was resumed by another process, using the \$RESUME system service or the \$SET PROCESS/RESUME DCL command.
HIB	The process was created in a hibernation state or the process itself calls the \$HIBER system service.	Either another process issued a \$WAKE system service or a wake up was scheduled, using the \$SCHDWK system service.
LEF	The process is waiting for an asynchronous event, such as an I/O (\$QIO), lock (\$ENQ), or timer (\$SETIMR).	An event flag was set, usually by the completion of an asynchronous event.
CEF	The process is waiting for a common event to be set, usually by another process that has associated with the same common event cluster (\$ASCEFC system service). Common event flags are used for interprocess synchronization.	Another process sets the common event flag using the \$SETEF system service.

Involuntary Wait States

State	Reason to Enter	Reason to Leave
PFW	The process has incurred a hard page fault, and it enters this state while waiting for the I/O to complete. Processes entering this state for an extended period of time may indicate a thrashing problem because of memory exhaustion.	The hard fault is resolved.
FPG	This state is entered when the free page list has been exhausted. This may indicate that too small a setting for FREELIM has been made.	A free page has been added to the free page list.
COLPG	A process has incurred a hard fault on a page that already has a hard fault incurred on it. This state most commonly occurs when processes accessing global sections are also incurring hard faults.	The hard fault is resolved.
MWAIT	This state occurs when a process is waiting on a mutual exclusion semaphore to be granted for system synchronization or when a system resource is not currently available. This state is interpreted by many system displays as MUTEX or RWxyz, where xyz identifies the resource on which the processes are waiting.	The MUTEX is granted, or the resource is made available.

States of Concern

- **COM**—A large number of computable (**COM**) processes may indicate a CPU bottleneck.

This bottleneck can be resolved by using faster CPU(s), using more CPUs (**SMP** configuration), having better algorithms, relegating CPU-bound tasks to lower priority (1-3) batch queues or, possibly lowering the **QUANTUM** **SYSGEN** parameter. With a **QUANTUM** setting of 20 (the default) in units of 10-ms intervals, five computable processes can force the next process to wait up to 1 second before the CPU is granted to it. A lower **QUANTUM** will spread the CPU cycles more evenly.

- **PFW**—This state indicates that hard paging is occurring.
You should consider using more memory or lowering the **WSQUOTAs** for processes. See memory management for more on paging problems.
- **COLPG**—This is the same as **PFW** except that the hard faults are most likely occurring on global pages.
- **FPG**—Memory is being depleted very rapidly. Consider using a higher **FREELIM** setting and purchasing more memory.
- **MUTEX** and miscellaneous resource wait—Determine the resource that is being blocked, and attempt to free it up.
- Any outswapped state—Determine whether **BALSETCNT** is set too low. This could cause artificial swapping.

MONITOR States

```

OpenVMS Monitor Utility
PROCESS STATES
on node ELLIS
8-SEP-1999 05:50:10.21

0      10      20      30      40
+ - - - - + - - - - + - - - - + - - - - +
Collided Page Wait      |
Mutex & Misc Resource Wait |
Common Event Flag Wait  |
Page Fault Wait         2 |**
Local Event Flag Wait    |
Local Evt Flg (Outswapped) 5 |*****
                        |      |      |      |
Hibernate                5 |*****
Hibernate (Outswapped)   9 |*****
Suspended                 |
Suspended (Outswapped)   |
Free Page Wait           |
Compute                  1 |*
Compute (Outswapped)     1 |*
Current Process          1 |*
                        + - - - - + - - - - + - - - - +

```

```

OpenVMS Monitor Utility
PROCESS STATES
on node ELLIS
8-SEP-1999 05:50:36.53

0      10      20      30      40
+ - - - - + - - - - + - - - - + - - - - +
Collided Page Wait      |
Mutex & Misc Resource Wait |
Common Event Flag Wait  |
Page Fault Wait         |
Local Event Flag Wait    |
Local Evt Flg (Outswapped) 3 |***
                        |      |      |      |
Hibernate                4 |****
Hibernate (Outswapped)  10 |*****
Suspended                 |
Suspended (Outswapped)   |
Free Page Wait           |
Compute                  1 |*
Compute (Outswapped)     5 |*****
Current Process          1 |*
                        + - - - - + - - - - + - - - - +

```

Module Outline

Objectives

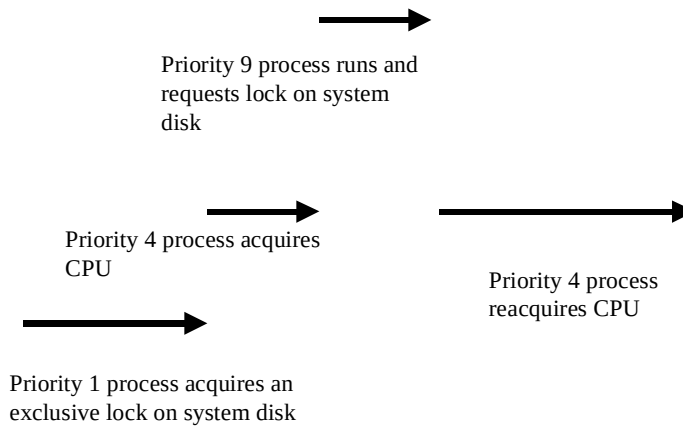
This module describes the OpenVMS method of preventing low priority processes from blocking high priority processes from running. This blocking is due to resource ownership not being released.

Topics

- PIXSCAN
- DORMANTWAIT

PIXSCAN and DORMANTWAIT

A mechanism must be implemented to help resolve the potential situation in which a high-priority process is being blocked for access to a resource by a low-priority process, which in turn owns the resource and is being blocked from the CPU by a midrange priority process.



Example Illustrating the Need for PIXSCAN

- To prevent this problem, the system checks every second to determine whether there are computable, dormant processes on the system.
 - A dormant process is one that has been computable for **DORMANTWAIT** seconds (defaults to 2) and has not received the CPU.
 - The system checks lowest priority computable queues and works up to higher priority queues, until either a process has been found that is not dormant or **PIXSCAN** processes have been found.
 - When a dormant process has been found, its priority is raised to the highest priority of any computable or current process on the system.
- In the case described in the previous diagram, the priority 1 process would have had its priority raised to 4 within **DORMANTWAIT*2** seconds and, it is hoped, would have acquired the CPU long enough to release the owned resource.

**Important**

- The system and **AUTOGEN** set these parameters at reasonable settings. Any modification of these parameters may lead to severe response time degradation.
-

Module Outline

Objectives

This module describes the OpenVMS requirement for locking and its impact on performance.

Topics

- Locking Concepts
- Other Characteristics of Locks
- Application Design and Locking Performance
- Interpreting the \$MONITOR LOCK Display

Locking Concepts

The need for locking:

Process A	Process B	Process C
Program 1	Program 2	Program 1
File X		

- When a process, while executing a program, attempts to access a record within a file that may be shared by other processes, we must guarantee that the record is in a consistent state when it is accessed.
 - To guarantee consistency, a lock is taken out on the record.
 - Locks are acquired through the **SY\$ENQ** system service.
 - The lock is acquired on a named resource.
- In this case, the lock would include some description of the record. When the process is done with the record, it releases the owned lock by calling the **SY\$DEQ** system service.
 - These system service calls are performed transparently to the application by the RMS and the XQP, although the application may use these services explicitly.
- Locks are required for consistent access to a shared resource, and the overhead associated with them must be written off as necessary system CPU time.
 - The time for locks is processed in kernel mode.
 - So you will have to write off some of your CPU time in kernel mode to lock processing.
 - Remember that unless you have determined that a database is being accessed for read-only access, locking is important and necessary to guarantee consistent data.
- The Lock Manager allows programmers to synchronize access to data within a cluster.

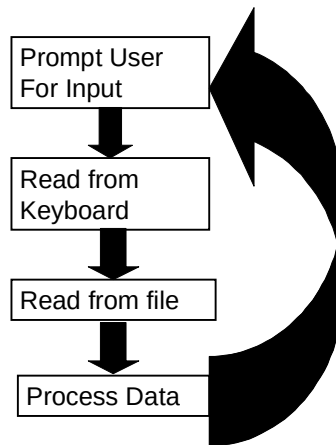
Other Characteristics of Locks

- Locks allow for multiple concurrent access to a shared resource.
 - Generally, there can be multiple readers but only single writers of a locked database.
 - Locks are acquired by mode, allowing additional granularity on the lock requests.
 - The lock modes include null, concurrent read, concurrent write, protected read, protected write, and exclusive.
- Null locks are always granted and are used primarily for lock conversions.
 - A lock conversion allows a lock to be converted to a more-protected or less-protected mode.
 - This conversion technique has advantages over using a **\$ENQ/DEQ** pair.
 - First, it preserves information that might describe changes to the resource.
 - This information is maintained in the lock value block.
 - Second, the conversion can be slightly faster than an **\$ENQ/DEQ** pair.
- Locks have built-in deadlock detection.
 - If a process owns resource A and will not release it until it acquires resource B, and at the same time, another process owns B and wants A, we could not resolve this situation, and the processes would wait forever.
 - If this situation is detected, the lock manager will allow it to continue for **DEADLOCKWAIT** (defaults to 10) seconds.
 - After this time has expired, the lock manager will choose a victim, and the process will be notified of the deadlock situation.
 - At this time, the process must release all of its owned locks and reacquire them to break the deadlock.
 - Deadlocks are an indication of a poorly designed application.
 - If you were to develop a hierarchy in which locks were acquired, such as alphabetically by primary key, the deadlocks could be prevented.
- Locks also support sublocks, which allow for a convenient naming convention, because resource names must be in a sublock hierarchy.
 - For example, record locks are sublocks of file locks.
 - Locks also function clusterwide, allowing for easy file sharing transparent to the applications that are accessing the files.

Application Design and Locking Performance

- RMS takes out locks implicitly when the process accesses files that have been opened for shared read/write access.
 - Two types of locking are supported for accessing a file stream through RMS: manual and automatic locking.
 - A stream to a file is typically established when a file is opened in a high-level language.
 - This open statement translates into an RMS **SYSS\$OPEN** and **SYSS\$CONNECT** call.
 - You can establish more than one stream through connect statements on an open file, if the file has been opened with multistreaming enabled.
 - This technique can be useful when you are performing sequential and random access to the same file.
- Automatic locking is the default on a file open.
 - When a file is automatically locked, the lock on the record stream is held until the next operation is performed on the record stream, the file is closed, or the record lock is explicitly unlocked by using a high-level construct or a call to the RMS routine **SYSS\$FREE** or **SYSS\$RELEASE**.
- Manual locking must be chosen by the programmer.
 - This technique is typically used when many records in a file form a transaction.
 - You would not want to release a lock on a given record until the transaction has been completed.
 - In this case, a lock is acquired on a record within a stream and is held until it has been specifically unlocked by using a high-level language construct or the RMS routine **SYSS\$FREE** or **SYSS\$RELEASE**, or until the file has been closed.

- The following diagram illustrates a potential problem in an application design using shared files.



Application Design Error with Locking

- In the above example, note that when the record has been read from the file, a "protected write" lock is taken out.
 - This is because RMS assumes that you are reading with the intent to write.
 - The record lock is held as long as the user remains at the terminal prompt, which blocks other users from accessing the locked record.
- In application design, when you are only reading the record and not intending to write it, you might consider taking out a protected read lock on the record.
 - Also, when you are going into a potentially extended wait period, you should release all held locks to prevent blocking other processes from accessing previously accessed records.
 - To take out a protected read lock on a record, check your high-level language or use a user open to access the record access block (**RAB**) on the record stream.
 - In the **RAB**, you should set the bit **RAB\$V_REA** and clear the bit **RAB\$V_RLK** in the field **RAB\$V_ROP**.

Locking Considerations

- Take great care when using manual locking to guarantee that all owned locks are given up properly.
 - Locking in a haphazard fashion can lead to deadlocks.
 - Holding onto locks for an extended period of time can cause deadlock searches, which are unproductive and extend the time that other processes wait for locks.
 - These behaviors may be difficult to detect.
- Most high level languages (**COBOL**, **FORTRAN**, **BASIC**, ...) do not support direct methods for enabling waits on locked records.
 - **FORTRAN** and **BASIC** allow the **RAB** to be manipulated. **COBOL** does not.

Record Options in RAB for Locking Shared Files

- The following bits can be set in the **Record Access Block (RAB)** to enable various locking modes:

Bit Position	Function
RAB\$V_NLK	Do not lock records (not functionally a correct definition—see lock mode table).
RAB\$V_NXR	Hold lock even if record accessed randomly is deleted. Only valid for relative files.
RAB\$V_REA	Read lock, allowing multiple readers and no writers.
RAB\$V_RLK	Record lock, allowing only a single concurrent writer.
RAB\$V_RRL	Read regardless of lock. If record is currently locked, alternate status is RMSS_OK_RRL .
RAB\$V_WAT	If record is locked, wait until unlocked or time-out.
RAB\$V_TMO	Enable time-out period for wait. RAB\$B_TMO contains time-out value.
RAB\$V_ULK	Enable manual locking.

- Lock modes for shared files based on bit combinations:

Bit Combinations	Lock Mode
REA=0 and RLK=0	Exclusive
RLK=1	Protected Write
REA=1 and RLK=0	Protected Read
NLK=1	Concurrent Read

Sample \$MONITOR LOCK/Local Buffers

OpenVMS Monitor Utility
 LOCK MANAGEMENT STATISTICS
 on node ELLIS
 3-APR-1999 04:02:30.06

	CUR	AVE	MIN	MAX
New ENQ Rate	384.00	384.00	384.00	384.00
Converted ENQ Rate	569.00	569.00	569.00	569.00
DEQ Rate	384.00	384.00	384.00	384.00
Blocking AST Rate	0.00	0.00	0.00	0.00
ENQs Forced To Wait Rate	62.33	62.33	62.33	62.33
ENQs Not Queued Rate	0.00	0.00	0.00	0.00
Deadlock Search Rate	0.00	0.00	0.00	0.00
Deadlock Find Rate	0.00	0.00	0.00	0.00
Total Locks	169.00	169.00	169.00	169.00
Total Resources	155.00	155.00	155.00	155.00

\$MONITOR LOCK Display (Global Buffers)

OpenVMS Monitor Utility
LOCK MANAGEMENT STATISTICS
on node ELLIS
3-APR-1999 03:59:50.03

	CUR	AVE	MIN	MAX
New ENQ Rate	565.33	565.33	565.33	565.33
Converted ENQ Rate	297.66	297.66	297.66	297.66
DEQ Rate	565.33	565.33	565.33	565.33
Blocking AST Rate	66.00	66.00	66.00	66.00
ENQs Forced To Wait Rate	129.66	129.66	129.66	129.66
ENQs Not Queued Rate	0.00	0.00	0.00	0.00
Deadlock Search Rate	0.00	0.00	0.00	0.00
Deadlock Find Rate	0.00	0.00	0.00	0.00
Total Locks	194.00	194.00	194.00	194.00
Total Resources	182.00	182.00	182.00	182.00

Interpreting the \$MONITOR LOCK Display

Rate	Description
New ENQ Rate Converted ENQ Rate DEQ Rate	These rates are a normal indication of locking due to the access of shared files. Little can be done with these numbers other than to write off any associated CPU time, based on the primitive times mentioned earlier. You might also want to determine whether files that are only being read have been opened for shared read-only access, eliminating the need for locks.
Blocking AST Rate	This number will go up for two primary reasons: Job controller processes on systems in a OpenVMS Cluster are attempting to communicate with one another, and processes are attempting to access buckets for which there was previously a lock held. The blocking AST is used as a signaling mechanism, along with an exclusive lock with an associated value block to identify changes to existing records. Little can be done with this number other than to write off the CPU time.
ENQs Forced To Wait Rate	If this number becomes high and you are accessing shared RMS files, it is probably normal activity associated with accessing buckets for shared files. If you are doing locking other than RMS-related locking, this number might imply that there is contention for shared resources. The application should be analyzed to allow for minimizing this contention.
ENQs Not Queued Rate	This number will typically go up when applications are using high-level languages to access shared files and there is contention for the shared records. The application should be analyzed for minimization of shared resource contention; that is, locks should be released before extended wait states.
Deadlock Search Rate Deadlock Find Rate	If the deadlock search rate goes high, but the deadlock find rate stays low, this is an indication that processes are using manual locking, or its equivalent through an application other than RMS, and holding onto locks for too long a period of time. The application should be examined and redesigned to minimize the duration for which locks are being held.
Total Locks	Locks are located on the system through a lock identification table. This table is sized by the SYSGEN parameter LOCKIDTBL at bootstrap time. If more locks ever exist than the size of this parameter, the table will grow by another LOCKIDTBL entry. This will cause a one-time performance hit. To minimize this hit, LOCKIDTBL should be at least as large as the maximum number of locks that have appeared on the system. The cost for over-sizing the table is 4 bytes of nonpaged pool for each unused entry in the table.
Total Resources	The resources on which you lock are located by name and hashing through a resource hash table. To obtain the most optimal hash, the table size should be the smallest prime number such that the table is 60 to 80 percent full. The cost for over-committing entries to this table is 4 bytes of nonpaged pool per entry. The table is sized by the SYSGEN parameter RESHASHTBL .

Lock Manager Parameters

Relevant **SYSGEN** parameters:

Parameter	Description
LOCKIDTBL	Initial size of lock ID table
LOCKIDTBL_MAX	Maximum size of lock ID table (obsolete)
RESHASHTBL	Size of resource hash table
DEADLOCK_WAIT	Amount of time to allow a potential deadlock to stall

\$ mcr sysgen

SYSGEN> SH LOCKDIRWT

Parameter Name	Current	Default	Min.	Max.	Unit	Dynamic
LOCKDIRWT	0	0	0	255	Pure-numbe	

SYSGEN> SH LOCKIDTBL

Parameter Name	Current	Default	Min.	Max.	Unit	Dynamic
LOCKIDTBL	1792	3840	1792	16776959	Entries	
LOCKIDTBL_MAX	38400	16776959	1792	16776959	Obsolete	

SYSGEN> SH RESHASHTBL

Parameter Name	Current	Default	Min.	Max.	Unit	Dynamic
RESHASHTBL	2048	64	1	16777216	Entries	

SYSGEN> EXIT

\$

The number of locks per process (**ENQLM**) are limited.

Distributed Locking Concepts

Module 26

Module Outline

Objectives

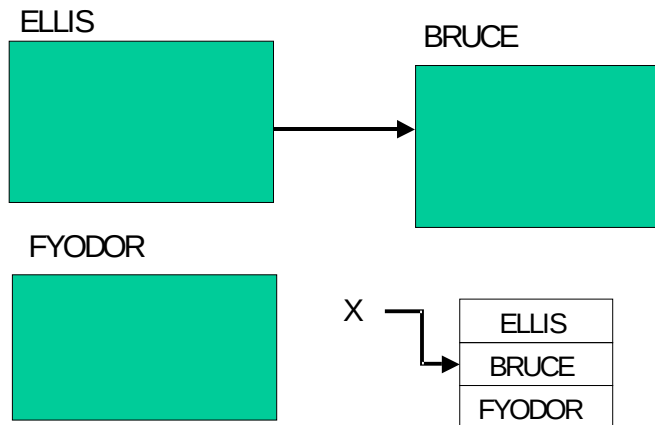
This module describes the performance characteristics associated with distributed locking. Upon completion of this module students will be able to determine and affect resource mastership in a OpenVMS Cluster.

Topics

- Distributed Locking Concepts
- The LOCKDIRWT SYSGEN Parameter
- Performance Issues with Distributed Locking

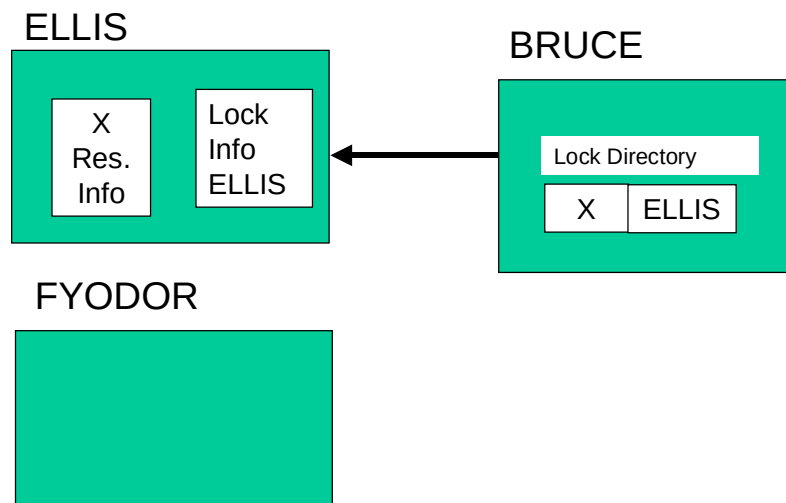
Distributed Locking Concepts

- Before discussing the performance implications of OpenVMS Cluster distributed locking, we must understand how distributed locking works.
 - Consider the scenario in which node **ELLIS** is interested in a resource named **X**, and there are two other nodes in the cluster, **BRUCE** and **FYODOR**.
 - Before performing any locking operations, **ELLIS** uses a hashing function on resource **X**.
 - For now, consider that the input to the hashing function will produce an index into a table identifying any one of the three nodes.
 - Let us say that **X** hashes to **BRUCE**, and then **BRUCE** has been determined to be the directory node for the resource named **X**.
 - **ELLIS** sends a message to the node **BRUCE** asking who is mastering resource **X**.



ELLIS Requests Notification of Mastership of Resource X

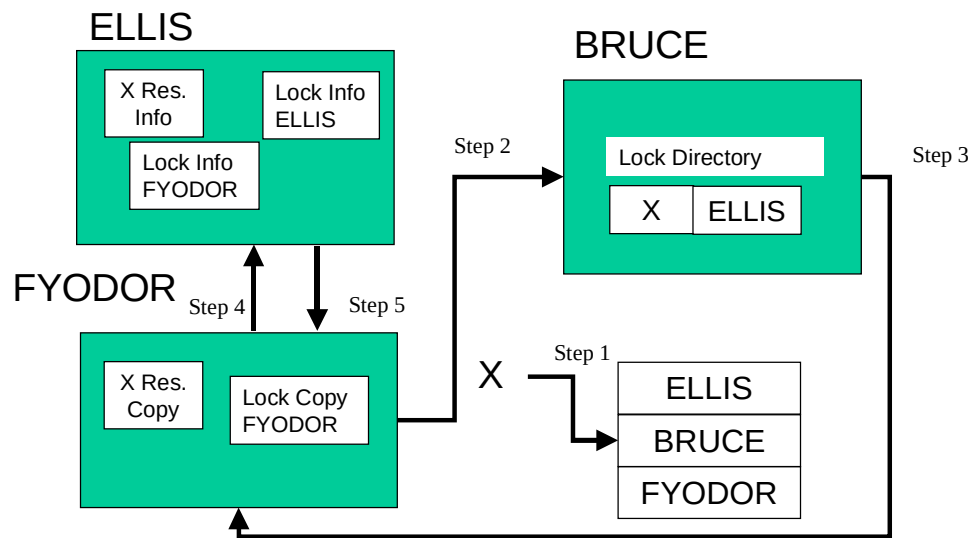
- Node **BRUCE** looks into its directory and sees that there is no entry.
 - So **BRUCE** makes a directory entry for **X** and identifies **ELLIS** as the resource master of **X**.
 - **ELLIS** maintains the primary resource information on **X**, and all conversions will be performed locally by **ELLIS**.
 - No additional wire transfers are required until **ELLIS** dequeues **X**.
 - Then a message must be sent to **BRUCE** telling it that the directory information should be dropped.



Mastership Is Granted

- The lock mastership scheme is interest based.
 - Because **ELLIS** was the first node interested in resource **X**, it became the resource master of **X**.
 - Had **FYODOR** been interested first, it would be the resource master.

- Let us assume that **FYODOR** becomes interested in resource **X** at the same time that **ELLIS** is mastering it.
 - It would hash to the same directory node, **BRUCE**, and ask about mastership.
 - **BRUCE** would identify that **ELLIS** was resource master, and **FYODOR** would send a lock request to **ELLIS** requesting access to the owned resource.
 - **ELLIS** would identify to **FYODOR** whether the lock was available.
 - **FYODOR** would now have to communicate with **ELLIS** each time it converted or dequeued the given lock request.



Determining the Resource Master for Disk Volumes

- While the process on **FYODOR** is waiting for information from the directory or master node, it appears in a **RWSCS** variety of **MWAIT** state.
 - These states will become noticeable if the node mastering resources is tied up processing on the interrupt stack.
 - The additional interrupt stack processing can be due to excessive mastering of resources, **MSCP** serving disks, or other activities.

LOCKDIRWT SYSGEN Parameter

The **LOCKDIRWT SYSGEN** parameter identifies the lock directory weighting scale for a given node in an OpenVMS Cluster.

Its meaning has changed over the versions of OpenVMS and can have various effects on locking, depending on the version of OpenVMS that you are running.

Meaning of LOCKDIRWT Before OpenVMS V4.6

In its simplest form, the **LOCKDIRWT SYSGEN** parameter before OpenVMS V4.6 identified how many entries a given node would have in the lock directory hashing table.

- If the setting for it was higher, there was a better chance of the given node being hashed to on a directory lookup.
- In these versions of OpenVMS, **LOCKDIRWT** generally should have been set corresponding to the speed ratings of a single CPU on a given node.

OpenVMS V4.6 to V4.7 Meaning of LOCKDIRWT

In OpenVMS V4.6 through V4.7, the same basic meaning was maintained for **LOCKDIRWT** as with earlier versions of OpenVMS, except when the node had a setting of 0 for **LOCKDIRWT**.

- In this case, a node would be "reluctant" to master resources; that is, it would master resources only when no other node was interested in the given resource.
- Because resource mastership did not move dynamically, this had meaning only during an OpenVMS Cluster state transition.
- In this case, mastership might have moved away from the node with the 0 **LOCKDIRWT** setting, as long as at least two other nodes in the OpenVMS Cluster had a non-zero **LOCKDIRWT** setting.
- This allowed mastership of locks to be biased away from particularly slow nodes.

OpenVMS V5.0 to V5.2 Meaning of LOCKDIRWT

In OpenVMS V5.0 through V5.2, the setting of **LOCKDIRWT** at 0 had the same previous meanings plus one other.

- If **LOCKDIRWT** was set to 0, then during an OpenVMS Cluster state transition, we could perform either:
 - No rebuild at all—when the node with **LOCKDIRWT** set to 0 was joining the OpenVMS Cluster.
 - A partial lock rebuild—when the node with **LOCKDIRWT** set to 0 was leaving the cluster.
- This allowed for minimizing the OpenVMS Cluster state transition time on nodes with **LOCKDIRWT** set to 0.

OpenVMS V5.3 and Later Meaning of LOCKDIRWT

With OpenVMS V5.3, mastership of a resource never moved unless all interest in the resource was given up.

- This was true even if the master node lost interest in the resource.
- Under V5.3 of OpenVMS, when the node mastering the resource loses interest in the resource, mastership moves to another node, on the basis of the setting for **LOCKDIRWT**, lock activity, and the parameter **PE1**.

PE1

The movement of a sublock hierarchy is dependent on the special **SYSGEN** parameter **PE1**. You might want to set negative if interrupt state/stack time is excessive. You can monitor movement with SDA **SHOW LOCK/SUMMARY**.

PE1 Setting	Behavior
0	Always move sublock tree.
PE1>0	Move tree if height is < PE1, based on formula using number of local vs. remote lock requests.
PE1<0	Never move tree.

SDA Lock Statistics Example

SDA> show lock/summ

Lock Manager Summary Information:

Lock Manager Poolzone:

Poolzone Region Address:	FFFFFFFF.80D8C380	
Packet Size:	00000100	(256.)
Number of Pages:	00000076	(118.)
Maximum Number of Pages:	000026B9	(9913.)
Free Page Count:	00000011	(17.)
Hits:	00000FAB	(4011.)
Misses:	00000038	(56.)
Poolzone Expansions:	00000076	(118.)
Allocation Failures:	00000000	(0.)
Allocation not from 1st Page:	00000000	(0.)
Empty Pages:	00000000	(0.)

Press RETURN for more.

SDA>

Lock Manager per-CPU Performance Counters:

Counters \ CPU Id	0	Total
-----	-----	-----
LKB Allocations (cache)	26726	26726
RSB Allocations (cache)	27960	27960
New Lock Requests (local)	27133	27133
New Lock Requests (in)	1832	1832
New Lock Requests (out)	620	620
Conversion Requests (loc)	10469	10469
Conversion Requests (in)	919	919
Conversion Requests (out)	14	14
Dequeue Requests (local)	25909	25909
Dequeue Requests (in)	704	704
Dequeue Requests (out)	621	621
\$ENQ Requests that Wait	77	77
\$ENQ Requests not Queued	13	13
Blocking ASTs (local)	18	18
Blocking ASTs (in)	1	1
Blocking ASTs (out)	2	2
Directory Functions (in)	3575	3575
Directory Functions (out)	1703	1703

Deadlock Counters:

Deadlock Searches	0
Deadlock Found	0
Deadlock Messages (in)	0
Deadlock Messages (out)	0

Lock Remaster Counters:	
Tree moved to this node	2
Tree moved to another node	14
Tree moved due to higher Activity	14
Tree moved due to higher LOCKDIRWT	0
Tree moved due to Single Node Locks	13
No Quota for Operation	1
Proposed New Manager Declined	0
Operations completed	16
Remaster Messages Sent	58
Remaster Messages Received	22
Remaster Rebuild Messages Sent	418
Remaster Rebuild Messages Received	289
2-Phase Commit Counters:	
Requests Sent	0
Requests Received	3
Ready Messages Sent	3
Ready Messages Received	0
ACK Messages Sent	0
ACK Messages Received	3
Cancel Messages Sent	0
Cancel Messages Sent	0
SDA>	

Performance Issues with Distributed Locking

- Generally, **LOCKDIRWT** should be set on the basis of the speed ratings of the systems (single CPU) in a **CI**-based OpenVMS Cluster.
- In an **LAVc**, the **LOCKDIRWT** setting involves a trade-off.
 - If your primary goal is faster OpenVMS Cluster state transition time, set it to 0 on the satellites and to a non-zero setting on the boot members.
 - However, if you notice processes on the satellites showing up in **RWSCS** states, and your goal is better overall performance, consider setting it exactly the opposite, that is, 0 on the boot members and non-zero on the satellites.
 - This setting will free up the boot members' interrupt stack time for **MSCP**-serving disks and spread the distributed locking processing across the satellites, which are generally initiating the lock requests.
 - This will be particularly effective if you have many satellites and your boot members are not significantly faster than the satellites.
- When a node boots into the OpenVMS Cluster, it will acquire a lock on the disk that it mounts.
 - This lock is requested every time a file is opened up.
 - If one node boots first and this node is particularly slow, it may slow down other nodes in the OpenVMS Cluster.
 - In this case, you might want to play with the booting order or the order in which disks are mounted in order to offload the overhead of mastering the device locks in a OpenVMS Cluster.
 - To determine which device locks are being mastered by a given node, enter the following commands:

```
$DEFINE/USER SHOW$DEBUG ELLIS  
$SHOW DEVICE/MOUNT
```
- In the next example, you see a **\$MONITOR PROCESS** display issued from the node **FRODO**.
 - A few processes are showing up in **RWSCS** state.

- The following example shows the display from **\$MONITOR DLOCK** on node **FRODO**.
 - In this case, most of the requests are outgoing; that is, they are being serviced by another node.
 - Although this is not illustrated, there was another node that was processing these requests and also mastering the device locks.
 - This node was incurring 50+ percent of its CPU time on the interrupt stack, blocking it from servicing these lock requests and stalling the processes on **FRODO** in **RWSCS** state.

Distributed Locking Example

Process Count: 30

VAX/VMS Monitor Utility

Uptime: 0 12:30:03

PROCESSES

on node FRODO

16-FEB-1990 01:56:19

PID	STATE	PRI	NAME	PAGES	DIOCNT	FAULTS	CPU TIME
21200081	HIB	16	SWAPPER	0/0	0	0	00:00:19.8
21200086	HIB	8	ERRFMT	0/101	384	76	00:00:08.2
21200087	HIB	16	CACHE_SERVER	0/94	76	63	00:00:00.7
21200088	HIB	10	CLUSTER_SERVER	20/232	3	124	00:00:20.0
21200089	HIB	7	OPCOM	0/212	130	370	00:00:18.3
2120008A	HIB	10	JOB_CONTROL	0/457	19611	259	00:08:19.1
2120008B	HIB	10	CONFIGURE	0/161	8	108	00:00:03.7
2120008C	HIB	9	SMISERVER	267/589	42	247	00:00:02.3
2120008F	LEF	6	uucp	3/69	170	3994	00:00:15.0
21200090	HIB	9	NETACP	0/415	11	228	00:01:12.1
21200091	HIB	6	EVL	61/307	24	3823	00:00:06.7
21200092	HIB	6	uxqt	10/77	129	4002	00:00:09.9
21200093	HIB	11	REMACP	0/42	2	69	00:00:00.2
21200094	LEF	6	uxqt dcl	77/248	2	261	00:00:01.4
21200096	LEF	11	XYP_SYMBIONT	32/400	1584	269	00:00:08.3
212000A2	LEF	6	M11CAL_ALARM	95/270	684	136	00:00:13.7
212000A5	LEF	15	RDMS_MONITOR	0/49	11	636	00:00:01.8
212000A7	HIB	13	RPC\$SWL	12/213	8	162	00:00:00.5
21200329	RWSCS	5	BATCH_3556	99/1549	779567	3152	00:21:21.1
212000AB	HIB	7	pcLINK_ACP	6/82	11	217	00:00:04.5
212000AC	LEF	5	PCLINK_VO	49/2036	1225	1861	00:01:09.7
212000AD	HIB	6	PCLINK_ADMIN	29/142	88	3681	00:00:42.4
212000B5	HIB	6	SYMBIONT_0001	2/63	144	4754	00:01:06.9
212000B6	HIB	6	SYMBIONT_0002	2/65	524	10806	00:05:20.0
212000B7	HIB	5	SYMBIONT_0003	31/512	1266	60218	01:03:19.4
21200238	RWSCS	5	BATCH_3653	100/1266	58907	3176	00:09:32.4
212000BA	HIB	9	SYMBIONT_0004	0/45	7	199	00:00:01.0
212000BC	HIB	9	SYMBIONT_0005	2/71	56	1877	00:00:04.1
2120013E	HIB	5	I/O Express	0/63	1032	3187	00:00:12.5
212002BF	CUR	5	Bruce Ellis	193/564	88	4283	00:01:10.1

VAX/VMS Monitor Utility
 DISTRIBUTED LOCK MANAGEMENT STATISTICS
 on node FRODO
 16-FEB-1990 01:56:16

		CUR	AVE	MIN	MAX
New ENQ Rate	(Local)	7.12	7.12	7.12	7.12
	(Incoming)	0.00	0.00	0.00	0.00
	(Outgoing)	13.35	13.35	13.35	13.35
Converted ENQ Rate	(Local)	7.71	7.71	7.71	7.71
	(Incoming)	0.00	0.00	0.00	0.00
	(Outgoing)	6.52	6.52	6.52	6.52
DEQ Rate	(Local)	7.12	7.12	7.12	7.12
	(Incoming)	0.00	0.00	0.00	0.00
	(Outgoing)	13.05	13.05	13.05	13.05
Blocking AST Rate	(Local)	0.00	0.00	0.00	0.00
	(Incoming)	0.00	0.00	0.00	0.00
	(Outgoing)	0.00	0.00	0.00	0.00
Dir Functn Rate	(Incoming)	0.00	0.00	0.00	0.00
	(Outgoing)	13.35	13.35	13.35	13.35
Deadlock Message Rate		0.00	0.00	0.00	0.00

VAX/VMS Monitor Utility
 DISTRIBUTED LOCK MANAGEMENT STATISTICS
 on node FRODO
 16-FEB-1990 01:56:19

		CUR	AVE	MIN	MAX
New ENQ Rate	(Local)	7.12	7.12	7.12	7.12
	(Incoming)	0.00	0.00	0.00	0.00
	(Outgoing)	12.75	13.05	12.75	13.35
Converted ENQ Rate	(Local)	7.71	7.71	7.71	7.71
	(Incoming)	0.00	0.00	0.00	0.00
	(Outgoing)	6.52	6.52	6.52	6.52
DEQ Rate	(Local)	7.12	7.12	7.12	7.12
	(Incoming)	0.00	0.00	0.00	0.00
	(Outgoing)	13.05	13.05	13.05	13.05
Blocking AST Rate	(Local)	0.00	0.00	0.00	0.00
	(Incoming)	0.00	0.00	0.00	0.00
	(Outgoing)	0.00	0.00	0.00	0.00
Dir Functn Rate	(Incoming)	0.00	0.00	0.00	0.00
	(Outgoing)	13.05	13.20	13.05	13.35
Deadlock Message Rate		0.00	0.00	0.00	0.00

Module Outline

Objectives

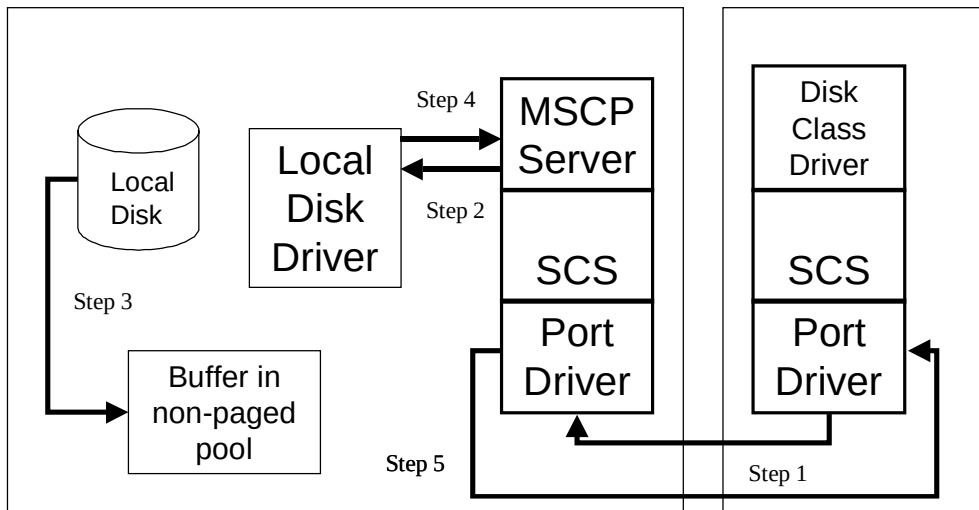
This module describes the effect of MSCP serving disks on CPU performance.

Topics

- MSCP Server
- Interpreting \$MONITOR MSCP

MSCP Server

- **MSCP** server allows disks that are locally connected in a VMSccluster to be served by or made available to other nodes in the cluster.
 - The server operates by receiving a request, such as a read from another node, and issuing the request to the local disk driver.
 - The data is read into buffers in nonpaged pool and then is transferred across the wire through the **MSCP** server.
 - Other processing is also performed by the system communication services (**SCS**) and the port drivers.



MSCP Server Operation

- Although MSCP serving is an I/O-related activity, its main cost is in CPU time spent on the interrupt stack required to service the given requests.
 - Bear in mind that when many simultaneous requests are being processed by the server, **MSCP** serving can use up a fairly large amount of CPU time.
 - Also keep in mind that this time is charged against the interrupt stack and is prohibitive.
- Other issues in **MSCP** serving include the management of the I/O buffers that are maintained in nonpaged pool.
 - These buffers are sized by the **SYSGEN** parameter **MSCP_BUFFER**.
 - This parameter defines the number of pages set aside for holding data coming in or going out through the server.
 - The more buffer space available, the less likely requests will be stalled because of waiting for buffers.
 - One indication that the buffers might need to be increased is displayed in the **\$SHOW DEVICE/SERVED** command.
 - If the buffer wait rate goes up - that is, the maximum hits 2 - you may consider increasing the **MSCP_BUFFER** parameter and nonpaged pool accordingly, to minimize buffer contention.
- Another indication that you should consider increasing the **MSCP_BUFFER** **SYSGEN** parameter is a fragmented request.
 - This is an indication that the contiguous available space for buffers is being fragmented and larger transfers must be broken into smaller requests, yielding poorer I/O performance.

VAX/VMS Monitor Utility
MSCP SERVER STATISTICS
on node THORIN
4-JAN-1999 12:53:22

		CUR	AVE	MIN	MAX
Server I/O Request Rate		1.32	1.32	1.32	1.32
Read Request Rate		1.32	1.32	1.32	1.32
Write Request Rate		0.00	0.00	0.00	0.00
Extra Fragment Rate		0.00	0.00	0.00	0.00
Fragmented Request Rate		0.00	0.00	0.00	0.00
Buffer Wait Rate		0.00	0.00	0.00	0.00
Request Size Rates	1	0.66	0.66	0.66	0.66
(Blocks)	2-3	0.00	0.00	0.00	0.00
	4-7	0.33	0.33	0.33	0.33
	8-15	0.33	0.33	0.33	0.33
	16-31	0.00	0.00	0.00	0.00
	32-63	0.00	0.00	0.00	0.00
	64+	0.00	0.00	0.00	0.00

Interpreting \$MONITOR MSCP

The following table shows the key fields in \$MONITOR MSCP.

Rate	Description
Extra Fragment Rate	This rate defines the number of fragments that are being required to complete transfers. If this number goes up, it may indicate a need for a larger buffer area (MSCP_BUFFER).
Fragmented Request Rate	This rate defines the number of requests that had to be broken into one or more additional fragments before the request could be completed. It is also an indication of a need for a larger buffer area.
Buffer Wait Rate	This is an indication that even with fragmenting buffers, enough contiguous buffer space could not be obtained. It is an indication of a need for a larger buffer area (MSCP_BUFFER).

The number and size of transfers are important in determining the nature of your interrupt stack time. Larger transfers take up more interrupt stack time.

Module Outline

Objectives

The OpenVMS system provides a variety of mechanisms for tailoring both the user interface to the system and the performance and functional characteristics of the system. This module describes the use and general functionality of the INSTALL utility and the SYSGEN utility.

Topics

- INSTALL Utility
- SYSGEN Utility
- AUTOGEN Overview

INSTALL Utility

The **INSTALL** utility can be used to make executable files known to the system and enhance the functional and/or performance characteristics of the system.

- These characteristics include:
 - The ability to eliminate I/Os associated with image activations, through maintaining information in virtual memory.
 - The ability to eliminate I/Os associated with memory management, through allowing sharing of code and possibly data pages or forcing code pages to be memory resident.
 - Allowing users to run with enhanced privileges on selected images.
 - The ability to track accounting information on selected images.

The following table summarizes the functional/performance characteristics that may be modified using the **INSTALL** utility:

Qualifier	Characteristic	Costs/Considerations
None	The image's file ID is kept memory resident, eliminating I/O costs associated with directory lookup at image activation.	10s of bytes of paged pool.
/OPEN	The image remains permanently open, until explicitly de-installed, typically at shutdown. This eliminates the I/O costs associated with opening the image at image activation.	100s of bytes of non-paged pool.
/HEADER_RESIDENT	The image's header is virtually memory resident, eliminating the need to read the image header at image activation time.	100s of bytes of paged pool, plus /OPEN costs.
/SHARED	The image is mapped through global pages, allowing for the image's read-only pages to be shared by multiple processes, without tying up unnecessary memory. Note, memory residence of these pages is not guaranteed. Data pages may be shared, using the /WRITE qualifier.	GBLSECTIONS sysgen parameter may need to be increased. Small cost of physical memory required to map global page table. On a VAX (and Alpha PreV7.1), the GBLPAGES sysgen parameter may need to be increased. The costs associated with /OPEN are implied.
/PRIVILEGED= (priv-list)	The image runs with enhanced privileges, which are taken away at image rundown.	The image must be linked /NOTRACEBACK.
/PROTECTED	The image is a privileged shareable image, meaning that it contains privileged mode (Kernel and Executive) system services.	There are several setup and linking requirements for these images. The costs are similar to those associated with /SHARE, as it must also be installed /SHARE.
/RESIDENT	Guarantees that the read-only code sections stay physically memory resident	EXPENSIVE. Should only be used on images that are used frequently. Costs 1 page of physical memory for every page tied up by image. NOTE: image must be linked with /SECTION_BINDING and there must be sufficient reserved granularity hint pages (SYSGEN parameter GH_RSRVPGCNT).
/EXECUTE_ONLY	Allows a shareable image that allows EXECUTE and no READ permissions, to be mapped at run-time.	Only for shareable images.
/ACCOUNTING	Enables image accounting for activations of this image.	More I/O activity to accounting file, but less than globally enabling image accounting.

- The **INSTALL** utility may be used interactively or at the DCL command prompt.
- When running interactively you can use **EXIT** or **^Z** to exit the session.
- Using the **INSTALL** utility:
 - `INSTALL HELP command`
 - `INSTALL ADD file-spec[/qualifier]`
 - `INSTALL DELETE file-spec`
 - `INSTALL LIST [file-spec]`
 - `INSTALL REPLACE file-spec[/qualifier]`
- The **INSTALL** utility will add a file, using the **ADD** command and any associated qualifiers will define the properties of the known image. The **INSTALL** utility will search the **SYSS\$SYSTEM** directory for the stated image file.

INSTALL Utility Example

- Example of listing known file information:

```
$ install list directory/full
```

```
DISK$AXP71FT2:<SYS0.SYSCOMMON.SYSEXE>.EXE
  DIRECTORY;1      Open Hdr Shar
    Entry access count      = 2
    Current / Maximum shared = 0 / 2
    Global section count    = 10
```

- Determining shared image resource utilization:

```
$ install list/global/summ
```

```
Summary of Memory Global Sections
```

```
399 Global Sections Used, 129008/47136 Global Pagelets Used/Unused
```

```
$
```

- Attempt to install SDA image fails, because it is already installed. You must use the REPLACE command to change the characteristics:

```
$ install add sda/share
```

```
%INSTALL-E-FAIL, failed to CREATE entry for
```

```
DISK$AXP71FT2:<SYS0.SYSCOMMON.SYSEXE
```

```
>SDA.EXE
```

```
-INSTALL-E-EXISTS, Known File Entry for a version of this file already exists
```

```
$ install list sda/full
```

```
DISK$AXP71FT2:<SYS0.SYSCOMMON.SYSEXE>.EXE
```

```
SDA;1
```

```
Entry access count      = 3
```

```
$
```

```
$ install replace sda/share
```

```
$ install list sda/full
```

```
DISK$AXP71FT2:<SYS0.SYSCOMMON.SYSEXE>.EXE
```

```
SDA;1      Open      Shar
```

```
Entry access count      = 0
```

```
Current / Maximum shared = 0 / 0
```

```
Global section count    = 5
```

SYSGEN Utility

The OpenVMS system has a set of parameters that are used to configure the layout of tables/memory and its run-time characteristics. These parameters are referred to as **SYSGEN** parameters.

- The **SYSGEN** parameters are maintained in binary form in parameter files, usually of file type **.PAR**. The default parameters used when the system boots are stored in the file **SYS\$SYSTEM:ALPHAVMSSYS.PAR** (on an ALPHA platform) or **SYS\$SYSTEM:VAXVMSSYS.PAR** (on a VAX platform).
- The **SYSGEN** utility reads parameters into a work environment, using the **USE** command. No modifications are actually made until a **WRITE** command is issued.
- **SYSGEN** parameters may be dynamic or non-dynamic. Dynamic parameters may be changed on the running system by issuing a **WRITE ACTIVE** command. **SYSGEN** does an implicit **USE ACTIVE** on entry.
- To change non-dynamic parameters, you must set the parameter values, perform a **WRITE CURRENT** and reboot the system.

Using SYSGEN

- Use one of the following commands to enter **SYSGEN**:
 - \$ RUN SYS\$SYSTEM:SYSGEN
 - \$ MCR SYSGEN
- To save the current settings:
SYSGEN> WRITE SYS\$SYSTEM:OLD_PARAMS.DAT
- To examine a parameter:
SYSGEN> SHOW QUANTUM
- To examine all parameters:
SYSGEN> SHOW/ALL
SYSGEN> SHOW/SPECIAL !Parameters which normally should not
 !be changed by customers.
- To examine dynamic parameters:
SYSGEN> SHOW/DYNAMIC
- To change a dynamic parameter:
SYSGEN> SET QUANTUM 15
SYSGEN> WRITE ACTIVE

Using SYSGEN/SYSMAN

Example of using SYSGEN:

```
$ mcr sysgen
SYSGEN> SH QUANTUM
Parameter Name      Current      Default      Min.      Max.      Unit Dynamic
-----
QUANTUM              20          20          2        32767    10Ms      D
SYSGEN>
SYSGEN> SET QUANTUM 15
SYSGEN> WRITE ACTIVE
SYSGEN> EXIT
$
```

- To change a nondynamic parameter:

```
SYSGEN> SET NPAGEDYN 6000000
```

```
SYSGEN> WRITE CURRENT
```

Then reboot.

- To get a handle on the meaning of a parameter:

```
SYSGEN> HELP PARAMETER MMG_CTLFLAGS
```

- To exit SYSGEN:

```
SYSGEN> EXIT
```

- Parameters may also be modified using the SYSMAN utility:

```
$ RUN SYS$SYSTEM:SYSMAN
```

```
SYSMAN> PARAMETER equivalent-SYSGEN-command
```

SYSMAN works the same way as SYSGEN, except the parameters may be modified across the cluster or network.

AUTOGEN

- The **SYSGEN** parameters have delicate interrelationships. It is possible to set these parameters in such a way as to render the system un-bootable.
- The recommended method for changing **SYSGEN** parameters is through the **AUTOGEN** command procedure, since **AUTOGEN** understands these interrelationships, as well as the general load on the system.
- **AUTOGEN** also gathers information about the system using feedback data, which is stored in system data cells, to generate an accurate picture of the system resource needs.
- **AUTOGEN** should generally be run when:
 - New hardware is added to the system, like memory.
 - When installing layered products.
 - When the system load changes.
 - When installing images **/SHARE** with the **INSTALL** utility.
- **AUTOGEN** resides in the directory **SYSS\$UPDATE**, while all of its data files reside in the directory **SYSS\$SYSTEM**.

USING AUTOGEN

AUTOGEN Command Format

@SYS\$UPDATE:AUTOGEN [start-phase] [end-phase] [execution-type]

- Start phases, normally, gather information from the system.
- End phases, normally, set up files to implement the parameter changes.
- Execution types include:

Type	Description
FEEDBACK	Force feedback information usage.
FEEDBACK	Ignore feedback information.
CHECK_FEEDBACK	Use feedback in calculations if reasonable.
None	Use feedback if reasonable, else abort before changes.

AUTOGEN Phases Details

Phase	Description	Input Files	Output Files
SAVPARAMS	Generates new feedback data	None	AGEN\$FEEDBACK.DAT
GETDATA	Generates the file PARAMS.DAT used to define new values for parameters	CLU\$PARAMS.DAT AGEN\$FEEDBACK.DAT MODPARAMS.DAT	PARAMS.DAT
GENPARAMS	Sets up commands to change parameters	PARAMS.DAT	SETPARAMS.DAT VMSIMAGES.DAT AGEN\$PARAMS.REPORT
TESTFILES	Determines appropriate sizes for page, swap, and dump files	PARAMS.DAT	AGEN\$PARAMS.REPORT
GENFILES	Changes sizes of page, swap, and dump files	PARAMS.DAT	PAGEFILE.SYS SWAPFILE.SYS SYSDUMP.DMP AGEN\$PARAMS.REPORT SWAPFILE
SETPARAMS	Sets parameters	SETPARAMS.DAT	ALPHAVMSSYS.PAR ALPHAVMSSYS.OLD
SHUTDOWN	Shuts system down		
REBOOT	Shuts system down and reboots, using new parameters.		

AUTOGEN Sample Usage

- Gather system information:
\$ @SYS\$UPDATE:AUTOGEN SAVPARAMS GENPARAMS
 - If you like it:
\$ @SYS\$UPDATE:AUTOGEN SETPARAMS REBOOT
 - If you like it, but **AUTOGEN** does not:
\$ @SYS\$UPDATE:AUTOGEN SAVPARAMS REBOOT FEEDBACK
- Otherwise edit **modparams.dat**.

MODPARAMS.DAT

- This is the only supported interface for user input to **AUTOGEN**.
- To use, edit **modparams.dat** adding the following parameter settings:

Parameter=value	Forced setting.
MIN_parameter=value	Absolute minimum.
MAX_parameter=value	Absolute maximum.
ADD_parameter=value	Adds to AUTOGEN calculated value.

- Sample lines:
WSMAX=8192
ADD_GBLSECTIONS=100
- Then run **AUTOGEN**.

AUTOGEN Example

Initial **AUTOGEN** run:

```
$ @sys$update:autozen savparams setparams
%AUTOGEN-I-BEGIN, SAVPARAMS phase is beginning.
%AUTOGEN-I-FEEDBACK, Gathering feedback
%AUTOGEN-I-NEWFILE, A new version of SYS$SYSTEM:AGEN$FEEDBACK.DAT
    has been created. You may wish to purge this file.
%AUTOGEN-I-END, SAVPARAMS phase has successfully completed.
%AUTOGEN-I-BEGIN, GETDATA phase is beginning.
%AUTOGEN-I-NEWFILE, A new version of SYS$SYSTEM:PARAMS.DAT has been created.
    You may wish to purge this file.
%AUTOGEN-I-END, GETDATA phase has successfully completed.
%AUTOGEN-I-BEGIN, GENPARAMS phase is beginning.

*****
%AUTOGEN-E-PHASE, Errors were detected with the FEEDBACK data.
    The ending phase is being changed to "TESTFILES" so that no
    modifications will take place without you first being able to
    review the results of feedback as reported in
    SYS$SYSTEM:AGEN$PARAMS.REPORT. If the results are OK, re-invoke
    AUTOGEN as follows:
        @SYS$update:autozen genparams setparams feedback
    If the results are NOT OK, you may wish to invoke AUTOGEN
    without using the feedback mechanism as follows:
        @SYS$update:autozen getdata setparams nofeedback
*****
%AUTOGEN-I-NEWFILE, A new version of SYS$MANAGER:VMSIMAGES.DAT has been
created.
    You may wish to purge this file.
%AUTOGEN-I-NEWFILE, A new version of SYS$SYSTEM:SETPARAMS.DAT has been
created.
    You may wish to purge this file.
%AUTOGEN-I-END, GENPARAMS phase has successfully completed.
%AUTOGEN-I-BEGIN, TESTFILES phase is beginning.

AUTOGEN calculations for page, swap, and dump files.

Page and Swap file calculations.

PAGEFILE1_SIZE information (for SYS$SYSTEM:PAGEFILE.SYS):
    Feedback information.
        Old value was 140000, New value is 84500
        Maximum observed usage: 56352
    PAGEFILE1_SIZE will be modified to hold 84500 blocks

SYS$SYSTEM will be 53% full when modified to hold 84500 blocks.
```

```

SWAPFILE1_SIZE information (for SYS$SYSTEM:SWAPFILE.SYS):
Feedback information.
Old value was 4500, New value is 6100
Maximum observed usage: 4096
SWAPFILE1_SIZE will be modified to hold 6100 blocks

```

```

SYS$SYSTEM will be 49% full when modified to hold 6100 blocks.
Errorlog dumpfile calculations:

```

```

No errorlog dump file modifications would have been made.
Errorlog dumpfile will remain at 34 blocks.

```

```

Dumpfile calculations:

```

```

A 65950 block dump file would have been created.

```

```

*****
%AUTOGEN-W-REPORT, Warnings were detected by AUTOGEN. Please review the
information given in the file SYS$SYSTEM:AGEN$PARAMS.REPORT
*****

```

```

%AUTOGEN-I-REPORT, AUTOGEN has produced some informational messages which
have been stored in the file SYS$SYSTEM:AGEN$PARAMS.REPORT. You may
wish to review the information in that file.
wish to review the information in that file.

```

```

%AUTOGEN-I-END, TESTFILES phase has successfully completed.

```

Determine size of page and swap file:

```

$ sh mem/fil
System Memory Resources on 16-JUN-1997 22:01:07.83

Paging File Usage (blocks):
DISK$AXP71FT2:[SYS0.SYSEXEX]SWAPFILE.SYS
384 384 4480
DISK$AXP71FT2:[SYS0.SYSEXEX]PAGEFILE.SYS
89536 50800 139904
$

```

Examine feedback report:

```

$ type sys$system:agen$params.report
AUTOGEN Parameter Calculation Report on node: ELLIS
This information was generated at 16-JUN-1997 21:54:38.52
AUTOGEN was run from SAVPARAMS to SETPARAMS - default execution specified

```

Processing Parameter Data files

Including parameters from: SYS\$SYSTEM:MODPARAMS.DAT

**** WARNING **** - The system was up for less than 24 hours (uptime = 12 hrs) when the feedback information was recorded. This could result in feedback information that does not accurately reflect your typical workload. However, this feedback information will be used in this run.

**** ERROR **** - Errors were detected with the FEEDBACK data.
The ending phase is being changed to "TESTFILES" so that no modifications will take place without you first being able to review the results of feedback as reported in SYS\$SYSTEM:AGEN\$PARAMS.REPORT. If the results are OK, re-invoke AUTOGEN as follows:

@SYS\$UPDATE:AUTOGEN GENPARAMS SETPARAMS FEEDBACK

If the results are NOT OK, you may wish to invoke AUTOGEN without using the feedback mechanism as follows:

@SYS\$UPDATE:AUTOGEN GETDATA SETPARAMS NOFEEDBACK

Feedback information was collected on 16-JUN-1997 21:54:37.94

Old values below are the parameter values at the time of collection.

The feedback data is based on 12 hours of up time.

Feedback information will be used in the subsequent calculations

Parameter information follows:

MAXPROCESSCNT parameter information:

Feedback information.

Old value was 35, New value is 34

Maximum Observed Processes: 24

Information on OpenVMS executable image Processing:

Processing SYS\$MANAGER:VMS\$IMAGES_MASTER.DAT

Total global pagelets counted = 44353

Total global sections counted = 124

Total resident code pages counted = 306

Total resident data pages counted = 0

GBLPAGFIL parameter information:

- AUTOGEN parameter calculation has been overridden.

The calculated value was 128. The value 1024

will be used in accordance with the following requirements:

GBLPAGFIL has been increased by 256.

GBLPAGFIL minimum value is 1024.

GBLPAGES parameter information:

Feedback information.

Old value was 176141, New value is 171597
Maximum used GBLPAGES: 129344
Global buffer requirements: 1024
Pagelets reserved for memory resident sections: 0

GBLSECTIONS parameter information:

Feedback information.

Old value was 600, New value is 600
Current used GBLSECTIONS: 400
Memory resident sections: 0

- AUTOGEN parameter calculation has been overridden.
The calculated value was 480. The value 600
will be used in accordance with the following requirements:
GBLSECTIONS minimum value is 600.

LOCKIDTBL parameter information:

Feedback information.

Old value was 1792, New value is 1792
Current number of locks: 153
Peak number of locks: 2048

RESHASHTBL parameter information:

Feedback information.

Old value was 2048, New value is 2048
Current number of resources: 136

MSCP_BUFFER parameter information:

Feedback information.

Old value was 128, New value is 128
MSCP server I/O rate: 0 I/Os per 10 sec.
I/Os that waited for buffer space: 0
I/Os that fragmented into multiple transfers: 0

SCSCONNCNT parameter information:

Feedback information.

Old value was 5, New value is 5
Peak number of nodes: 1
Number of CDT allocation failures: 0

SCSRESPCNT parameter information:

Feedback information.

Old value was 300, New value is 300
RDT stall count: 0

SCSBUFFCNT parameter information:

Feedback information.

Old value was 50, New value is 50
CIBDT stall count: 0

NPAGEDYN parameter information:

Feedback information.

Old value was 1474560, New value is 1589248

Maximum observed non-paged pool size: 1507328 bytes.

Non-paged pool request rate: 0 requests per 10 sec.

LNMSHASHHTBL parameter information:

Feedback information.

Old value was 512, New value is 512

Current number of shareable logical names: 363

ACP_DIRCACHE parameter information:

Feedback information.

Old value was 66, New value is 66

Hit percentage: 98%

Attempt rate: 6 attempts per 10 sec.

ACP_DINDXCACHE parameter information:

Feedback information.

Old value was 16, New value is 16

Hit percentage: 99%

Attempt rate: 5 attempts per 10 sec.

ACP_HDRCACHE parameter information:

Feedback information.

Old value was 66, New value is 66

Hit percentage: 89%

Attempt rate: 2 attempts per 10 sec.

ACP_MAPCACHE parameter information:

Feedback information.

Old value was 16, New value is 16

Hit percentage: 0%

Attempt rate: 0 attempts per 10 sec.

PAGEDYN parameter information:

Feedback information.

Old value was 622592, New value is 737280

Current paged pool usage: 436400 bytes.

Paged pool request rate: 0 requests per 10 sec.

GH_EXEC_CODE parameter information:

Feedback information.

Old value was 512, New value is 1024

GH_EXEC_DATA parameter information:

Feedback information.

Old value was 128, New value is 96

GH_RES_CODE parameter information:

Feedback information.

Old value was 1024, New value is 1024

GH_RES_DATA parameter information:

Feedback information.

Old value was 0, New value is 0

PROCSECTCNT parameter information:

- AUTOGEN parameter calculation has been overridden.

The calculated value was 32. The value 64

will be used in accordance with the following requirements:

PROCSECTCNT minimum value is 64.

PQL_MWSDEFAULT parameter information:

- AUTOGEN parameter calculation has been overridden.

The calculated value was 896. The value 1024

will be used in accordance with the following requirements:

PQL_MWSDEFAULT minimum value is 1024.

VAXCLUSTER parameter information:

- AUTOGEN parameter calculation has been overridden.

The calculated value was 1. The value 0

will be used in accordance with the following requirements:

VAXCLUSTER has been specified by a hard-coded value of 0.

Page, Swap, and Dump file calculations

will be used in accordance with the following requirements:

PROCSECTCNT minimum value is 64.

PQL_MWSDEFAULT parameter information:

- AUTOGEN parameter calculation has been overridden.

The calculated value was 896. The value 1024

will be used in accordance with the following requirements:

PQL_MWSDEFAULT minimum value is 1024.

VAXCLUSTER parameter information:

- AUTOGEN parameter calculation has been overridden.

The calculated value was 1. The value 0

will be used in accordance with the following requirements:

VAXCLUSTER has been specified by a hard-coded value of 0.

Page, Swap, and Dump file calculations

PAGEFILE1_SIZE information (for SYS\$SYSTEM:PAGEFILE.SYS):

Feedback information.

Old value was 140000, New value is 84500

Page and Swap file calculations:

PAGEFILE1_SIZE information (for SYS\$SYSTEM:PAGEFILE.SYS):

Feedback information.

Old value was 140000, New value is 84500

Maximum observed usage (blocks): 56352

PAGEFILE1_SIZE will be modified to hold 84500 blocks

SYS\$SYSTEM will be 53% full when modified to hold 84500 blocks.

SWAPFILE1_SIZE information (for SYS\$SYSTEM:SWAPFILE.SYS):

Feedback information.

Old value was 4500, New value is 6100

Maximum observed usage (blocks): 4096

SWAPFILE1_SIZE will be modified to hold 6100 blocks

SYS\$SYSTEM will be 49% full when modified to hold 6100 blocks.

Errorlog dumpfile calculations:

No errorlog dump file modifications would have been made.

Errorlog dumpfile will remain at 34 blocks.

Dumpfile calculations:

A 65950 block dump file would have been created.

\$

Old modparams.dat:

\$ type sys\$system:modparams.dat;-1

! SYS\$SYSDEVICE:[SYS0.SYSEXE]MODPARAMS.DAT

! Created during installation of OpenVMS AXP G7.1 28-MAY-1997 20:42:11.31

!

VAXCLUSTER=0

SCSNODE="ELLIS"

SCSSYSTEMID=1048

!

! End of SYS\$SYSDEVICE:[SYS0.SYSEXE]MODPARAMS.DAT

! Created during installation of OpenVMS AXP G7.1 28-MAY-1997 20:42:11.31

\$

Edited modparams.dat:

```
$ type sys$system:modparams.dat
! SYS$SYSDEVICE:[SYS0.SYSEXE]MODPARAMS.DAT
! Created during installation of OpenVMS AXP G7.1 28-MAY-1997 20:42:11.31
!
VAXCLUSTER=0
SCSNODE="ELLIS"
SCSSYSTEMID=1048
MIN_PAGEFILE1_SIZE=140000
MIN_SWAPFILE1_SIZE=6100
! Allow for new resident images
GH_RSRVPGCNT=20
!
! End of SYS$SYSDEVICE:[SYS0.SYSEXE]MODPARAMS.DAT
! Created during installation of OpenVMS AXP G7.1 28-MAY-1997 20:42:11.31
$
$
```

Change the parameters:

```

$ @sys$update:autogen savparams setparams feedback
%AUTOGEN-I-BEGIN, SAVPARAMS phase is beginning.
%AUTOGEN-I-FEEDBACK, Gathering feedback
%AUTOGEN-I-NEWFILE, A new version of SYS$SYSTEM:AGEN$FEEDBACK.DAT
    has been created. You may wish to purge this file.
%AUTOGEN-I-END, SAVPARAMS phase has successfully completed.
%AUTOGEN-I-BEGIN, GETDATA phase is beginning.
%AUTOGEN-I-NEWFILE, A new version of SYS$SYSTEM:PARAMS.DAT has been created.
    You may wish to purge this file.
%AUTOGEN-I-END, GETDATA phase has successfully completed.
%AUTOGEN-I-BEGIN, GENPARAMS phase is beginning.
%AUTOGEN-I-NEWFILE, A new version of SYS$MANAGER:VMSIMAGES.DAT has been
created.
    You may wish to purge this file.
%AUTOGEN-I-NEWFILE, A new version of SYS$SYSTEM:SETPARAMS.DAT has been created.
    You may wish to purge this file.
%AUTOGEN-I-END, GENPARAMS phase has successfully completed.
%AUTOGEN-I-BEGIN, GENFILES phase is beginning.
%SYSGEN-I-EXTENDED, SYS$SYSROOT:[SYSEXE]SWAPFILE.SYS;1 extended
%SYSGEN-I-EXTENDED, SYS$SYSROOT:[SYSEXE]SYSDUMP.DMP;1 extended

*****
%AUTOGEN-W-REPORT, Warnings were detected by AUTOGEN. Please review the
    information given in the file SYS$SYSTEM:AGEN$PARAMS.REPORT
*****

%AUTOGEN-I-REPORT, AUTOGEN has produced some informational messages which
    have been stored in the file SYS$SYSTEM:AGEN$PARAMS.REPORT. You may
    wish to review the information in that file.

%AUTOGEN-I-END, GENFILES phase has successfully completed.
%AUTOGEN-I-BEGIN, SETPARAMS phase is beginning.
%AUTOGEN-I-SYSGEN, parameters modified
%AUTOGEN-I-END, SETPARAMS phase has successfully completed.

```

Verify changes:

```

$ mcr sysgen
SYSGEN> SH GH_RSRV
Parameter Name      Current      Default      Min.      Max.      Unit Dynamic
-----
GH_RSRVPGCNT        0           0           0         2048      Pages
SYSGEN> USE CURR
SYSGEN> SH GH_RSRV
Parameter Name      Current      Default      Min.      Max.      Unit Dynamic
-----
GH_RSRVPGCNT        20          0           0         2048      Pages
SYSGEN> EXIT
$

```

Virtual Balance Slots

Appendix B

Module Outline

Objectives

This chapter describes virtual balance slots and their effect on VAX/VMS systems.

Topic

Virtual Balance Slots

Virtual Balance Slots

- Prior to version 6.0 the maximum number of concurrent memory resident processes was limited by system virtual address space.
- Virtual balance slots allow the number of concurrent memory resident processes to exceed **BALSETCNT**, within the limit of **MAXPROCESSCNT-2**. This mechanism is only implemented on VAX/VMS systems and not under Alpha.
- This is implemented through timesharing of the system virtual address space.
- A virtual balance slot holds the mapping for a memory resident process that does not currently own a real balance slot (RBS).
- The set of real balance slots are time shared among all memory resident processes.
- With VBS enabled, the quantity of memory resident processes is limited by the system parameter **MAXPROCESSCNT**.
- With VBS disabled, the quantity of memory resident processes is limited by the system parameter **BALSETCNT**.
- The system parameter **VBSS_ENABLE** enables virtual balance slots.
 - Bit 0 - Enables VBS. The default is disabled.
 - Bit 1 - Enables the creation of a map for process based direct I/O allowing the process with a direct I/O outstanding to be transitioned to a VBS. The default is enabled.
- **MONITOR VBS** displays information about the processing of virtual balance slots.
- **MONITOR PROC/TOPRBS** displays information about the process reuse of virtual balance slots.

\$ show system

VAX/VMS V6.1 on node BRUDEN 21-NOV-1994 06:34:17.09 Uptime 0 00:53:20

Pid	Process Name	State	Pri	I/O	CPU	Page flts	Ph.Mem	
00000021	SWAPPER	HIB	16	0	0 00:01:33.50	0	0	
00000082	_RTA1:	CUR	4	95	0 00:00:01.33	488	441	
00000063	THRASHER	RWMPB	6	3	0 00:00:34.60	135071	404	
00000164	H	COM	4	3	0 00:00:52.40	75	100	
00000025	IPCACP	HIB	10	9	0 00:00:00.14	221	73	
00000026	ERRFMT	HIB	8	61	0 00:00:00.48	153	105	
00000027	OPCOM	HIB	8	70	0 00:00:01.66	329	39	
00000028	AUDIT_SERVER	HIB	10	57	0 00:00:00.88	486	124	
00000029	JOB_CONTROL	HIB	8	76	0 00:00:00.49	321	116	
0000002A	SECURITY_SERVER	HIB	10	25	0 00:00:01.74	3251	792	
0000002B	TP_SERVER	HIB	8	225	0 00:00:04.99	2445	149	
0000014C	A	SUSP	4	3	0 00:00:29.00	76	101	
0000002D	DECW\$SERVER_0	HIB	6	1334	0 00:01:12.08	5615	827	
0000002E	DECW\$SESSION	LEF	6	412	0 00:00:12.21	6419	74	
000000EF	B	SUSP	4	2	0 00:00:19.14	76	101	
00000030	DECW\$MWM	LEF	4	172	0 00:00:10.35	3982	303	
00000031	VUE\$ELLIS_2	LEFO	5	--	swapped out --		463	S
00000032	DECW\$TE_0032	COM	4	177	0 00:02:30.45	4561	3151	
00000033	ELLIS	HIB	5	307	0 00:00:08.35	1368	159	
00000034	ELLIS_1	HIB	5	266	0 00:00:05.47	2397	371	S
00000035	ELLIS_2	LEF	9	3781	0 00:01:02.99	22549	270	S
00000036	_FTA4:	HIB	6	2068	0 00:00:35.86	3725	340	
00000057	C	SUSP	4	2	0 00:00:13.54	76	101	
00000058	D	SUSP	4	2	0 00:00:12.56	76	101	
00000059	E	SUSP	4	2	0 00:00:09.94	76	101	
0000003A	NETACP	HIB	9	50	0 00:00:00.78	245	235	
0000003B	EVL	HIB	6	54	0 00:00:00.74	569	66	N
0000003C	REMACP	HIB	9	17	0 00:00:00.16	102	57	
0000009D	F	SUSP	4	2	0 00:00:04.76	76	101	

\$ show memory/slots

System Memory Resources on 21-NOV-1994 06:34:26.97

Slot Usage (slots):	Total	Free	Resident	Swapped
Process Entry Slots	30	0	29	1
Balance Set Slots	19	1	18	0

\$ show memory

System Memory Resources on 21-NOV-1994 06:34:35.10

Physical Memory Usage (pages):	Total	Free	In Use	Modified
Main Memory (8.00Mb)	16384	2786	13063	535
Virtual I/O Cache Usage (pages):	Total	Free	In Use	Maximum
Cache Memory	46	5	41	8498
Slot Usage (slots):	Total	Free	Resident	Swapped
Process Entry Slots	30	0	29	1
Balance Set Slots	19	1	18	0
Dynamic Memory Usage (bytes):	Total	Free	In Use	Largest
Nonpaged Dynamic Memory	838144	141568	696576	78272
Paged Dynamic Memory	500224	313552	186672	311888
Paging File Usage (pages):	Free	Reservable	Total	
DISK\$BRUDEN:[SYS0.SYSEXE]SWAPFILE.SYS	2096	2096	2696	
DISK\$BRUDEN:[SYS0.SYSEXE]PAGEFILE.SYS	3555	-31938	14000	

Of the physical pages in use, 4876 pages are permanently allocated to VMS.

```

$ stop f
$
$ r/det/page=30000/proc=x thrasher
%RUN-S-PROC_ID, identification of created process is 000000BD
$ show sys
VAX/VMS V6.1 on node BRUDEN 21-NOV-1994 06:35:16.02 Uptime 0 00:54:19
  Pid   Process Name   State Pri   I/O      CPU      Page flts Ph.Mem
00000021 SWAPPER      HIB    16      0  0 00:01:42.82      0      0
00000082 _RTA1:         CUR     4    183  0 00:00:02.31     763    464
00000063 THRASHER     RWMPB   6      3  0 00:00:46.52    181530   404
00000164 H              COM     4      3  0 00:01:16.44      75    100
00000025 IPCACP       HIB    10      9  0 00:00:00.14     221     73
00000026 ERRFMT      HIB     8     61  0 00:00:00.48     153    105
00000027 OPCOM       HIB     8     70  0 00:00:01.66     329     39
00000028 AUDIT_SERVER HIB    10     57  0 00:00:00.88     486    124
00000029 JOB_CONTROL  HIB     9     77  0 00:00:00.49     321     87
0000002A SECURITY_SERVER HIB   10     25  0 00:00:01.79    3251    446
0000002B TP_SERVER    HIB     9    229  0 00:00:05.09    2520    148
0000014C A          SUSP    4      3  0 00:00:29.00      76    101
0000002D DECW$SERVER_0 HIB     6   1334  0 00:01:13.29    5785    791
0000002E DECW$SESSION LEF     6    412  0 00:00:12.23    6427     66
000000EF B          SUSP    4      2  0 00:00:19.14      76    101
00000030 DECW$MWM      LEF     4    172  0 00:00:10.35    3982    303
00000031 VUE$ELLIS_2    LEFO    5      -- swapped out --      463  S
00000032 DECW$TE_0032 COM     4    177  0 00:02:32.57    4561   3151
00000033 ELLIS        HIB     5    307  0 00:00:08.35    1368    159
00000034 ELLIS_1       HIB     5    266  0 00:00:05.47    2397    371  S
00000035 ELLIS_2       LEF     9   3781  0 00:01:02.99   22549    270  S
00000036 _FTA4:      HIB     7   2159  0 00:00:36.01    3735    201
00000057 C          SUSP    4      2  0 00:00:13.54      76    101
00000058 D          SUSP    4      2  0 00:00:12.56      76    101
00000059 E          SUSPO   4      -- swapped out --      101
0000003A NETACP      HIB     9     50  0 00:00:00.78     245    235
0000003B EVL         HIB     5     54  0 00:00:00.75     574     54  N
0000003C REMACP      HIB     9     17  0 00:00:00.16     102     57
000000BD X          RWMPB   6      3  0 00:00:01.34    3749   1733
$

```


\$ ana/sys

VAX/VMS System analyzer

SDA> show summ

Current process summary

```

-----
Extended Indx Process name   Username   State   Pri   PCB       PHD       Wkset
-- PID --
00000021 0001 SWAPPER                HIB      16 806953A0 80695200      0
000000A2 0002 H                        ELLIS    SUSP     9 8082C880 80E5E600    101
00000083 0003 I                        ELLIS    SUSP     4 8082D1C0 00000000    101
00000184 0004 J                        ELLIS    SUSP     9 80834180 80F01200    101
00000025 0005 IPCACP                SYSTEM   HIB     10 80822F80 80BD3600     73
00000026 0006 ERRFMT                SYSTEM   HIB     8 80823180 80ECAE00     93
00000027 0007 OPCOM                  SYSTEM   HIB     8 80823380 80C09A00    121
00000028 0008 AUDIT_SERVER          AUDIT$SERVER HIB     10 80823580 80CAC600     93
00000029 0009 JOB_CONTROL            SYSTEM   HIB     8 80823780 80E28200     80
0000002A 000A SECURITY_SERVER        SYSTEM   HIB     10 80823B80 80D18E00    723
0000002B 000B TP_SERVER              SYSTEM   HIB     10 80823D80 80D4F200    148
0000016C 000C K                        ELLIS    SUSP     4 80835BC0 00000000    101
0000002D 000D DECW$SERVER_0          ELLIS    HIB     8 80824180 80DBBA00    573
0000002E 000E DECW$SESSION          ELLIS    LEF     6 80823F80 80DF1E00     66
0000010F 000F A                        ELLIS    SUSP     4 80836600 00000000    101
00000030 0010 DECW$MWM              ELLIS    LEF     4 80822400 80D85600    347

```

Virtual Balance Slots Press RETURN for more.

SDA> set proc i

SDA> show proc

Process index: 0003 Name: I Extended PID: 00000083

Status : 40100801 res,suspen,login,softsus

Status2: 00000021 quantum_resched,vbsres

```

PCB address      8082D1C0      JIB address      80832900
PHD address      00000000      Swapfile disk address 00000000
Master internal PID 00040003      Subprocess count      0
Internal PID     00040003      Creator internal PID   00000000
Extended PID     00000083      Creator extended PID   00000000
State            SUSP          Termination mailbox    0000
Current priority 4              AST's enabled          KESU
Base priority    4              AST's active           S
UIC              [00020,000001]  AST's remaining        100
Mutex count      0              Buffered I/O count/limit 100/100
Waiting EF cluster 0          Direct I/O count/limit 100/100
Starting wait time 1B001B16      BUFI0 byte count/limit 32576/32576
Event flag wait mask 80758DA0      # open files allowed left 99
Local EF cluster 0 E0000000      Timer entries allowed left 8
Local EF cluster 1 00000000      Active page table count 0
Global cluster 2 pointer 00000000  Process WS page count 101
Global cluster 3 pointer 00000000  Global WS page count 0

```

SDA> show proc/phd

Process index: 0003 Name: I Extended PID: 00000083

Process header

First free P0 address	00000600	Accumulated CPU time	000002BD
Free PTEs between P0/P1	48867	CPU since last quantum	FFFB
First free P1 address	7FF6DA00	Subprocess quota	8
Free page file pages	32099	AST limit	100
Page fault cluster size	64	Process header index	000F
Page table cluster size	2	Backup address vector	000012D9
Flags	008E	WSL index save area	00001200
Direct I/O count	0	PTs having locked WSLs	6
Buffered I/O count	2	PTs having valid WSLs	8
Limit on CPU time	00000000	Active page tables	9
Maximum page file count	32768	Maximum active PTs	10
Total page faults	76	Guaranteed fluid WS pages	20
File limit	100	Extra dynamic WS entries	241
Timer queue limit	8	Locked WSLE counts array	522C
Current page file template 03000000		Valid WSLE counts array	53B3

Process page file assignments

PROCIDX	SYSIDX	REFCNT	
0	3	25	Current assignment
1	0	0	
2	0	0	
3	0	0	
Remaining reserved pages		27	Total reserved pages 103

SDA> **show summ**

Current process summary

```
-----
Extended Indx Process name  Username  State  Pri  PCB      PHD      Wkset
-- PID --
00000021 0001 SWAPPER          HIB     16 806953A0 80695200      0
000000A2 0002 H                ELLIS   SUSP     9 8082C880 80E5E600    101
00000083 0003 I                ELLIS   SUSP     9 8082D1C0 80F01200    101
00000184 0004 J                ELLIS   SUSP     9 80834180 00000000    101
00000025 0005 IPCACP          SYSTEM  HIB    10 80822F80 80BD3600     73
00000026 0006 ERRFMT          SYSTEM  HIB     8 80823180 80ECAE00     93
00000027 0007 OPCOM           SYSTEM  HIB     8 80823380 80C09A00    121
00000028 0008 AUDIT_SERVER    AUDIT$SERVER HIB    10 80823580 80CAC600     93
00000029 0009 JOB_CONTROL     SYSTEM  HIB     8 80823780 80E28200     80
0000002A 000A SECURITY_SERVER SYSTEM  HIB    10 80823B80 80D18E00    543
0000002B 000B TP_SERVER       SYSTEM  HIB    10 80823D80 80D4F200    148
0000016C 000C K                ELLIS   SUSP     4 80835BC0 00000000    101
0000002D 000D DECW$SERVER_0    ELLIS   HIB     8 80824180 80DBBA00    573
0000002E 000E DECW$SESSION    ELLIS   LEF     6 80823F80 80DF1E00     74
0000010F 000F A                ELLIS   SUSP     4 80836600 00000000    101
00000030 0010 DECW$MWM        ELLIS   LEF     4 80822400 80D85600    347
00000031 0011 VUE$ELLIS_2     ELLIS   LEFO    5 80822780 00000000    463
```

Press RETURN for more.

SDA>

SDA> **Exit**

\$

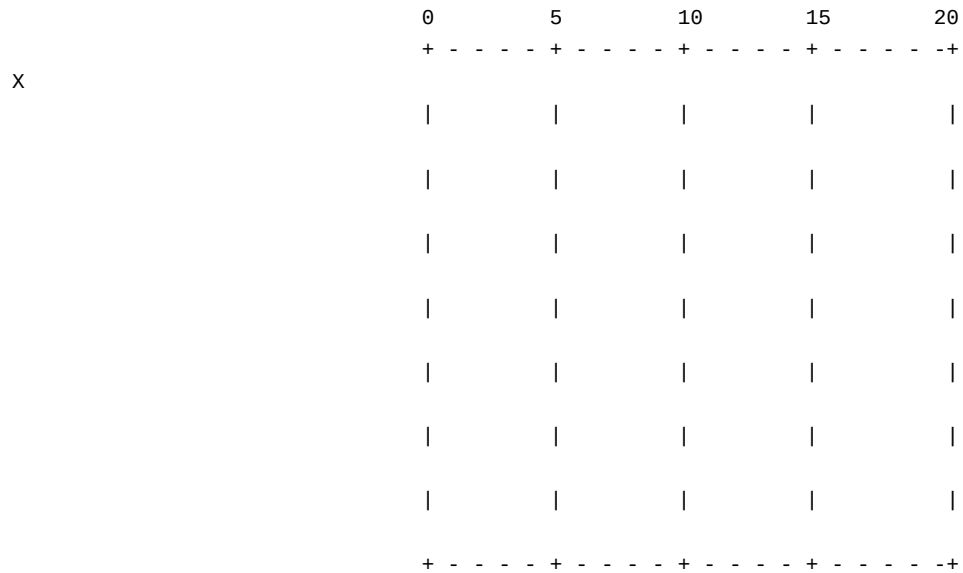
Monitoring Virtual Balance Slots

- An RBS (real balance slot) fault occurs when an attempt to access a real balance slot determines that the RBS is mapped through a VBS (virtual balance slot). The VBS is then moved into the RBS, causing CPU time to be incurred. Long waiting VBSs will be outswapped.
- Display from: **\$ MONITOR VBS, PROCESS/TOPRBS**

VAX/VMS Monitor Utility
 VIRTUAL BALANCE SLOT STATISTICS
 on node BRUDEN
 21-NOV-1994 06:31:55

	CUR	AVE	MIN	MAX
RBS Fault Rate	0.32	0.10	0.00	0.33
CPU Utilization	0.00	0.00	0.00	0.00

VAX/VMS Monitor Utility
 TOP RBS FAULT RATE PROCESSES
 on node BRUDEN
 21-NOV-1994 06:31:58



Module Outline

Objectives

This module presents the mechanisms used by Fast I/O to enhance I/O performance. Buffer objects, used to provide a target for I/O, are also presented. Students will be able to program using the **sys\$io_perform** system service and will be able to create buffer objects, in which to perform Fast I/O.

Topics

- Fast I/O
- Buffer Objects

Fast I/O

- Fast I/O is a performance enhanced alternative to performing QIOs.
- Fast I/O uses buffer objects to eliminate the I/O overhead of manipulating I/O buffers. General performance enhancements:
 - Buffer allocation/deallocation (BIO)
 - Locking down user buffers (DIO)
 - I/O post processing buffer cleanup
 - Implicitly eliminates some spinlock acquisition
- Performed using the buffer objects and the following system services:
sys\$io_setup
sys\$io_perform
sys\$io_cleanup

Buffer Objects

- Buffer objects:
 - Are similar to long term user buffer manipulation for direct I/O.
 - Use locked memory doubly mapped through process and system virtual address space.
 - Are physically memory resident!
 - Support 64-bit virtual address space buffers.
 - Are created through **sys\$create_bufobj** (*jacket*)
/sys\$create_bufobj_64
- System management considerations:
 - **MAXBOBMEM SYSGEN** parameter limits memory usage (defaults to 100)
 - **VMS\$BUFFER_OBJECT_USER** identifier is required for process.

Fast I/O Services Considerations

- Considerations when using the Fast I/O system services:
 - The I/O status area (**iosa**) replaces **iosb** (must be quadword aligned).
 - User buffers aligned on 512 byte boundaries.
 - Buffer and **iosa** must be passed through buffer objects.
 - Transfers **MUST** be in multiples of 512 bytes.
 - NO efn! Synchronization internal to **sys\$io_perform** or through ASTs.
- File handle:
 - The file handle or fandle is used to maintain context of the **sys\$io_perform** calls.

Implementing Fast I/O

Steps involved in performing fast I/O:

1. Assign a channel to the device.
2. Create the buffer object using **sys\$create_bufobj**.
3. Set up the future **io_perform** calls using **sys\$io_setup**.
4. Perform I/Os using **sys\$io_perform** until done.
 - Note synchronization is performed through ASTs or using synchronous form (**sys\$io_performw**).
 - Fastest if you use ASTs and wait on last I/O to complete.
 - AST communication through **IOSA\$IH_CONTEXT**.
5. Disassociate with **sys\$io_perform** using **sys\$io_cleanup** and delete the buffer object (**sys\$delete_bufobj**) (done implicitly on image rundown).

IO_perform/buffer object system services arguments:

```
sys$io_setup(func, bufobj, iosobj, astadr, flags, return_fandle)
```

```
sys$io_perform(fandle, chan, iosadr, bufadr, buflen, porint(p3))
```

```
sys$io_cleanup(fandle)
```

```
sys$create_bufobj_64(start_va_64, len_64, acmode, flags, retva_64,  
retlen_64, bufhandle_64)
```

```
sys$delete_bufobj_64(bufhandle_64)
```

