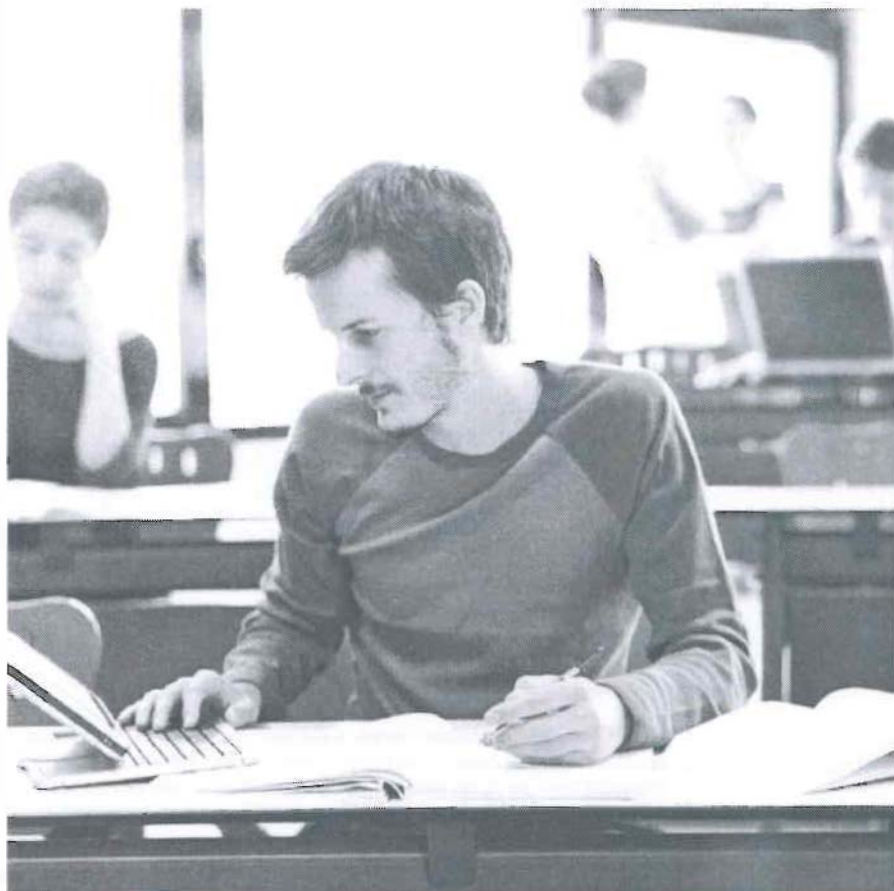




HP-UX Performance and Tuning
H4262S C.00

HP Training

Student guide



Copyright 2004 Hewlett-Packard Development Company, L.P.

The information contained herein is subject to change without notice. The only warranties for HP products and services are set forth in the express warranty statements accompanying such products and services. Nothing herein should be construed as constituting an additional warranty. HP shall not be liable for technical or editorial errors or omissions contained herein.

This is an HP copyrighted work that may not be reproduced without the written permission of HP. You may not use these materials to deliver training to any person outside of your organization without the written permission of HP.

UNIX® is a registered trademark of the Open Group.

Printed in the USA

HP-UX Performance and Tuning
Student guide
May 2004

Contents

Overview	1
Module 1 — Introduction	
1-1. SLIDE: Welcome to HP-UX Performance and Tuning.....	1-2
1-2. SLIDE: Course Outline	1-3
1-3. SLIDE: System Performance	1-4
1-4. SLIDE: Areas of Performance Problems.....	1-6
1-5. SLIDE: Performance Bottlenecks	1-8
1-6. SLIDE: Baseline	1-10
1-7. SLIDE: Queuing Theory of Performance	1-12
1-8. SLIDE: How Long Is the Line?.....	1-13
1-9. SLIDE: Example of Queuing Theory	1-14
1-10. SLIDE: Summary.....	1-16
1-11. LAB: Establishing a Baseline.....	1-17
1-12. LAB: Verifying the Performance Queuing Theory	1-19
Module 2 — Performance Tools	
2-1. SLIDE: HP-UX Performance Tools.....	2-2
2-2. SLIDE: HP-UX Performance Tools (Continued)	2-3
2-3. SLIDE: Sources of Tools.....	2-4
2-4. SLIDE: Types of Tools	2-6
2-5. SLIDE: Criteria for Comparing the Tools	2-8
2-6. SLIDE: Data Sources.....	2-10
2-7. SLIDE: Performance Monitoring Tools (Standard UNIX).....	2-11
2-8. TEXT PAGE: <code>iostat</code>	2-12
2-9. TEXT PAGE: <code>ps</code>	2-14
2-10. TEXT PAGE: <code>sar</code>	2-16
2-11. TEXT PAGE: <code>time</code> , <code>timex</code>	2-18
2-12. TEXT PAGE: <code>top</code>	2-19
2-13. TEXT PAGE: <code>uptime</code> , <code>w</code>	2-21
2-14. TEXT PAGE: <code>vmstat</code>	2-22
2-15. SLIDE: Performance Monitoring Tools (HP Specific)	2-25
2-16. TEXT: <code>glance</code>	2-26
2-17. TEXT PAGE: <code>gpm</code>	2-28
2-18. TEXT PAGE: <code>xload</code>	2-30
2-19. SLIDE: Data Collection Performance Tools (Standard UNIX)	2-31
2-20. TEXT PAGE: <code>acct</code> Programs	2-32
2-21. TEXT PAGE: <code>sar</code>	2-34
2-22. SLIDE: Data Collection Performance Tools (HP-Specific)	2-36
2-23. TEXT PAGE: MeasureWare/OVPA and DSI Software.....	2-37
2-24. TEXT PAGE: PerfView/OVPM	2-39
2-25. SLIDE: Network Performance Tools (Standard UNIX).....	2-41
2-26. TEXT PAGE: <code>netstat</code>	2-42
2-27. TEXT PAGE: <code>nfsstat</code>	2-44
2-28. TEXT PAGE: <code>ping</code>	2-46
2-29. SLIDE: Network Performance Tools (HP-Specific)	2-48
2-30. TEXT PAGE: <code>lanadmin</code>	2-49

Contents

2-31. TEXT PAGE: lanscan	2-51
2-32. TEXT PAGE: net tune (HP-UX 10.x Only)	2-53
2-33. TEXT PAGE: n dd (HP-UX 11.x Only)	2-55
2-34. TEXT PAGE: NetMetrix (HP-UX 10.20 and 11.0 Only)	2-57
2-35. SLIDE: Performance Administrative Tools (Standard UNIX)	2-58
2-36. TEXT PAGE: ipcs, ipcrm	2-59
2-37. TEXT PAGE: nice, renice	2-61
2-38. SLIDE: Performance Administrative Tools (HP-Specific)	2-63
2-39. Text Page: getprivgrp, setprivgrp	2-64
2-40. Text Page: rtprio	2-66
2-41. Text Page: rtsched	2-67
2-42. Text Page: scsictl	2-69
2-43. Text Page: serialize	2-71
2-44. Text Page: fsadm	2-72
2-45. Text Page: gettext, setext	2-74
2-46. Text Page: newfs, tune fs, vxtune fs	2-75
2-47. Text Page: Process Resource Manager (PRM)	2-77
2-48. Text Page: Work Load Manager (WLM)	2-78
2-49. Text Page: Web Quality of Service — WebQoS	2-79
2-50. SLIDE: System Configuration and Utilization Information (Standard UNIX)	2-80
2-51. TEXT PAGE: bdf, df	2-81
2-52. TEXT PAGE: mount	2-83
2-53. SLIDE: System Configuration and Utilization Information (HP-Specific)	2-84
2-54. TEXT PAGE: diskinfo	2-85
2-55. TEXT PAGE: dmesg	2-86
2-56. TEXT PAGE: ioscan	2-88
2-57. TEXT PAGE: vgdisplay, pvdisplay, lvdisplay	2-90
2-58. TEXT PAGE: swapinfo	2-92
2-59. TEXT PAGE: sysdef	2-93
2-60. TEXT PAGE: kmtune, kcweb	2-95
2-61. SLIDE: Application Profiling and Monitoring Tools (Standard UNIX)	2-96
2-62. TEXT PAGE: prof, gprof	2-97
2-63. Text page: Application Response Measurement (ARM) Library Routines	2-98
2-64. SLIDE: Application Profiling and Monitoring Tools (HP-Specific)	2-99
2-65. Text page: Transaction Tracker	2-100
2-66. Text page: caliper — HP Performance Analyzer	2-101
2-67. SLIDE: Summary	2-103
2-68. LAB: Performance Tools Lab	2-104

Module 3 — GlancePlus

3-1. SLIDE: This Is GlancePlus	3-2
3-2. SLIDE: GlancePlus Pak Overview	3-4
3-3. SLIDE: gpm and glance	3-6
3-4. SLIDE: glance — The Character Mode Interface	3-8
3-5. SLIDE: Looking at a glance Screen	3-11
3-6. SLIDE: gpm — The Graphical User Interface	3-13
3-7. SLIDE: Process Information	3-15
3-8. SLIDE: Adviser Components	3-17
3-9. SLIDE: adviser Bottleneck Syntax Example	3-18
3-10. SLIDE: The parm File	3-19

Contents

3-11.	SLIDE: GlancePlus Data Flow.....	3-21
3-12.	SLIDE: Key GlancePlus Usage Tips.....	3-23
3-13.	SLIDE: Global, Application, and Process Data	3-24
3-14.	SLIDE: Can't Solve What's Not a Problem.....	3-25
3-15.	SLIDE: Metrics: "No Answers without Data".....	3-26
3-16.	SLIDE: Summary.....	3-27
3-17.	SLIDE: HP GlancePlus Guided Tour	3-28
3-18.	LAB: gpm and glance Walk-Through	3-29

Module 4 — Process Management

4-1.	SLIDE: The HP-UX Operating System.....	4-2
4-2.	SLIDE: Virtual Address Process Space (PA-RISC)	4-4
4-3.	SLIDE: Virtual Address Process Space (IA-64)	4-6
4-4.	SLIDE: Physical Process Components.....	4-7
4-5.	SLIDE: The Life Cycle of a Process	4-9
4-6.	SLIDE: Process States.....	4-11
4-7.	SLIDE: CPU Scheduler	4-14
4-8.	SLIDE: Context Switching.....	4-16
4-9.	SLIDE: Priority Queues.....	4-17
4-10.	SLIDE: Nice Values.....	4-19
4-11.	SLIDE: Parent-Child Process Relationship.....	4-20
4-12.	SLIDE: glance — Process List.....	4-21
4-13.	SLIDE: glance — Individual Process.....	4-23
4-14.	SLIDE: glance — Process Memory Regions	4-24
4-15.	SLIDE: glance — Process Wait States.....	4-25
4-16.	LAB: Process Management.....	4-26

Module 5 — CPU Management

5-1.	SLIDE: Processor Module.....	5-2
5-2.	SLIDE: Symmetric Multiprocessing.....	5-4
5-3.	SLIDE: Cell Module	5-5
5-4.	SLIDE: Multi-Cell Processing	5-6
5-5.	SLIDE: CPU Processor	5-8
5-6.	SLIDE: CPU Cache	5-11
5-7.	SLIDE: TLB Cache	5-12
5-8.	SLIDE: TLB, Cache, and Memory	5-14
5-9.	SLIDE: HP-UX — Performance Optimized Page Sizes.....	5-16
5-10.	SLIDE: CPU — Metrics to Monitor Systemwide.....	5-19
5-11.	SLIDE: CPU — Metrics to Monitor per Process.....	5-21
5-12.	SLIDE: Activities that Utilize the CPU	5-23
5-13.	SLIDE: glance — CPU Report.....	5-25
5-14.	SLIDE: glance — CPU by Processor	5-26
5-15.	SLIDE: glance — Individual Process.....	5-27
5-16.	SLIDE: glance — Global System Calls	5-28
5-17.	SLIDE: glance — System Calls by Process	5-29
5-18.	SLIDE: sar Command.....	5-30
5-19.	SLIDE: timex Command	5-32
5-20.	SLIDE: Tuning a CPU-Bound System — Hardware Solutions	5-33
5-21.	SLIDE: Tuning a CPU-Bound System — Software Solutions.....	5-35
5-22.	SLIDE: CPU Utilization and MP Systems.....	5-36

Contents

5-23.	SLIDE: Processor Affinity	5-37
5-24.	LAB: CPU Utilization, System Calls, and Context Switches	5-38
5-25.	LAB: Identifying CPU Bottlenecks	5-40

Module 6 — Memory Management

6-1.	SLIDE: Memory Management	6-2
6-2.	SLIDE: Memory Management — Paging	6-4
6-3.	SLIDE: Paging and Process Deactivation	6-5
6-4.	SLIDE: The Buffer Cache	6-7
6-5.	SLIDE: The <code>syncer</code> Daemon	6-9
6-6.	SLIDE: IPC Memory Allocation	6-10
6-7.	SLIDE: Memory Metrics to Monitor — Systemwide	6-12
6-8.	SLIDE: Memory Metrics to Monitor — per Process	6-14
6-9.	SLIDE: Memory Monitoring <code>vmstat</code> Output	6-16
6-10.	SLIDE: Memory Monitoring <code>glance</code> — Memory Report	6-18
6-11.	SLIDE: Memory Monitoring <code>glance</code> — Process List	6-19
6-12.	SLIDE: Memory Monitoring <code>glance</code> — Individual Process	6-20
6-13.	SLIDE: Memory Monitoring <code>glance</code> — System Tables	6-21
6-14.	SLIDE: Tuning a Memory-Bound System — Hardware Solutions	6-23
6-15.	SLIDE: Tuning a Memory-Bound System — Software Solutions	6-24
6-16.	SLIDE: PA-RISC Access Control	6-26
6-17.	SLIDE: The <code>serialize</code> Command	6-28
6-18.	LAB: Memory Leaks	6-30

Module 7 — Swap Space Performance

7-1.	SLIDE: Swap Space Management — Simple View	7-2
7-2.	SLIDE: Swap Space — After a New Process Executes	7-4
7-3.	SLIDE: The <code>swapinfo</code> Command	7-5
7-4.	SLIDE: Swap Space Management — Realistic View	7-7
7-5.	SLIDE: Swap Space — After a New Process Executes	7-8
7-6.	SLIDE: Swap Space — When Memory Equals Data Swapped	7-10
7-7.	SLIDE: Swap Space — When Swap Space Fills Up	7-11
7-8.	SLIDE: Pseudo Swap	7-12
7-9.	SLIDE: Total Swap Space Calculation — with Pseudo Swap	7-14
7-10.	SLIDE: Example Situation Using Pseudo Swap	7-16
7-11.	SLIDE: Swap Priorities	7-17
7-12.	SLIDE: Swap Chunks	7-18
7-13.	SLIDE: Swap Space Parameters	7-19
7-14.	SLIDE: Summary	7-21
7-15.	LAB: Monitoring Swap Space	7-22

Module 8 — Disk Performance Issues

8-1.	SLIDE: Disk Overview	8-2
8-2.	SLIDE: Disk I/O — Read Data Flow	8-4
8-3.	SLIDE: Disk I/O — Write Data Flow (Synchronous)	8-6
8-4.	SLIDE: Disk Metrics to Monitor — Systemwide	8-8
8-5.	SLIDE: Disk Metrics to Monitor — Per Process	8-10
8-6.	SLIDE: Activities that Create a Large Amount of Disk I/O	8-12
8-7.	SLIDE: Disk I/O Monitoring <code>sar -d</code> Output	8-14
8-8.	SLIDE: Disk I/O Monitoring <code>sar -b</code> Output	8-16
8-9.	SLIDE: Disk I/O Monitoring <code>glance</code> — Disk Report	8-18

Contents

8-10.	SLIDE: Disk I/O Monitoring glance — Disk Device I/O	8-19
8-11.	SLIDE: Disk I/O Monitoring glance — Logical Volume I/O	8-20
8-12.	SLIDE: Disk I/O Monitoring glance — System Calls per Process	8-21
8-13.	SLIDE: Tuning a Disk I/O-Bound System — Hardware Solutions	8-22
8-14.	SLIDE: Tuning a Disk I/O-Bound System — Perform Asynchronous Meta-data I/O	8-24
8-15.	SLIDE: Tuning a Disk I/O-Bound System — Load Balance across Disk Controllers	8-26
8-16.	SLIDE: Tuning a Disk I/O-Bound System — Load Balance across Disk Drives	8-28
8-17.	SLIDE: Tuning a Disk I/O-Bound System — Tune Buffer Cache	8-30
8-18.	LAB: Disk Performance Issues	8-33

Module 9 — HFS File System Performance

9-1.	SLIDE: HFS File System Overview	9-2
9-2.	SLIDE: Inode Structure	9-5
9-3.	SLIDE: Inode Data Block Pointers	9-6
9-4.	SLIDE: How Many Logical I/Os Does It Take to Access /etc/passwd?	9-8
9-5.	SLIDE: File System Blocks and Fragments	9-10
9-6.	SLIDE: Creating a New File on a Full File System	9-13
9-7.	SLIDE: HFS Metrics to Monitor — Systemwide	9-15
9-8.	SLIDE: Activities that Create a Large Amount of File System I/O	9-17
9-9.	SLIDE: HFS I/O Monitoring bdf Output	9-18
9-10.	SLIDE: HFS I/O Monitoring glance — File System I/O	9-19
9-11.	SLIDE: HFS I/O Monitoring glance — File Opens per Process	9-20
9-12.	SLIDE: Tuning a HFS I/O-Bound System — Tune Configuration for Workload	9-22
9-13.	SLIDE: Tuning a HFS I/O-Bound System — Use Fast Links	9-25
9-14.	LAB: HFS Performance Issues	9-27

Module 10 — VxFS Performance Issues

10-1.	SLIDE: Objectives	10-2
10-2.	SLIDE: JFS History and Version Review	10-5
10-3.	SLIDE: JFS Extents	10-9
10-4.	SLIDE: Extent Allocation Policies	10-11
10-5.	SLIDE: JFS Intent Log	10-13
10-6.	SLIDE: Intent Log Data Flow	10-16
10-7.	SLIDE: Understand Your I/O Workload	10-18
10-8.	SLIDE: Performance Parameters	10-20
10-9.	SLIDE: Choosing a Block Size	10-21
10-10.	SLIDE: Choosing an Intent Log Size	10-23
10-11.	SLIDE: Intent Log Mount Options	10-25
10-12.	SLIDE: Other JFS Mount Options	10-27
10-13.	SLIDE: JFS Mount Option: mincache=direct	10-31
10-14.	SLIDE: JFS Mount Option: mincache=tmppcache	10-33
10-15.	SLIDE: Kernel Tunables	10-35
10-16.	SLIDE: Fragmentation	10-37
10-17.	TEXT PAGE: Monitoring and Repairing File Fragmentation	10-40
10-18.	SLIDE: Using setext	10-50
10-19.	SLIDE: I/O Tunable Parameters	10-52

Contents

10-20.	SLIDE: <code>vxtunefs</code> Command for Tuning VxFS	10-54
10-21.	SLIDE: <code>/etc/vx/tunefstab</code> Configuration	10-56
10-22.	SLIDE: Taking Snapshots and Performance	10-58
10-23.	LAB: JFS File System Tuning.....	10-60
 Module 11 — Network Performance		
11-1.	SLIDE: The OSI Model	11-2
11-2.	SLIDE: NFS Read/Write Data Flow	11-4
11-3.	SLIDE: NFS on HP-UX with UDP	11-6
11-4.	SLIDE: NFS on HP-UX with TCP	11-7
11-5.	SLIDE: <code>biod</code> on Client	11-9
11-6.	SLIDE: TELNET	11-11
11-7.	SLIDE: FTP	11-13
11-8.	SLIDE: Metrics to Monitor — NFS	11-15
11-9.	SLIDE: Metrics to Monitor — Network	11-18
11-10.	SLIDE: Determining the NFS Workload	11-20
11-11.	SLIDE: NFS Monitoring — <code>nfsstat</code> Output	11-23
11-12.	SLIDE: Network Monitoring — <code>lanadmin</code> Output	11-28
11-13.	SLIDE: Network Monitoring — <code>netstat -i</code> Output	11-31
11-14.	SLIDE: <code>glance</code> — NFS Report	11-32
11-15.	SLIDE: <code>glance</code> — NFS System Report.....	11-33
11-16.	SLIDE: <code>glance</code> — Network by Interface Report.....	11-34
11-17.	SLIDE: Tuning NFS	11-35
11-18.	SLIDE: Tuning the Network	11-37
11-19.	SLIDE: Tuning the Network (Continued).....	11-39
11-20.	LAB: Network Performance	11-41
 Module 12 — Tunable Kernel Parameters		
12-1.	SLIDE: Kernel Parameter Classes	12-2
12-2.	SLIDE: Tuning the Kernel.....	12-5
12-3.	SLIDE: Kernel Parameter Categories.....	12-8
12-4.	SLIDE: File System Kernel Parameters	12-9
12-5.	SLIDE: Message Queue Kernel Parameters	12-11
12-6.	SLIDE: Semaphore Kernel Parameters.....	12-13
12-7.	SLIDE: Shared Memory Kernel Parameters	12-15
12-8.	SLIDE: Process-Related Kernel Parameters	12-17
12-9.	SLIDE: Memory-Related Kernel Parameters.....	12-19
12-10.	SLIDE: LVM-Related Kernel Parameters	12-21
12-11.	SLIDE: Networking-Related Kernel Parameters.....	12-22
12-12.	SLIDE: Miscellaneous Kernel Parameters.....	12-23
 Module 13 — Putting It All Together		
13-1.	SLIDE: Review of Bottleneck Characteristics	13-2
13-2.	SLIDE: Performance Monitoring Flowchart	13-4
13-3.	SLIDE: Review — Memory Bottlenecks.....	13-6
13-4.	SLIDE: Correcting Memory Bottlenecks	13-7
13-5.	SLIDE: Review — Disk Bottlenecks	13-8
13-6.	SLIDE: Correcting Disk Bottlenecks.....	13-9
13-7.	SLIDE: Review — CPU Bottlenecks	13-11
13-8.	SLIDE: Correcting CPU Bottlenecks.....	13-12
13-9.	SLIDE: Final Review — Major Symptoms.....	13-13

Appendix A — Applying GlancePlus Data

A-1. TEXT PAGE: Case Studies — Using GlancePlus.....	A-2
--	-----

Solutions

Contents

Overview

Course Description

This course is intended to introduce students to the various aspects of monitoring and tuning their systems. Students will be taught how to monitor which tools to use, symptoms to look for, and what remedial actions to take. The course also covers HP GlancePlus/Gpm and HP PerfRx.

The course is designed to:

- Introduce the subject of performance and tuning.
- Describe how the system works.
- Identify what tools we can use to look at performance.
- Identify the symptoms we may encounter and what they indicate.

Course Goals

- To educate the students on HP-UX performance monitoring
- To enable them to identify bottlenecks and potential problems
- To learn the appropriate remedial actions to take

Student Performance Objectives

Module 1 — Introduction

- List characteristics of a system yielding good user response time.
- List characteristics of a system yielding high data throughput.
- List three generic areas most often analyzed for performance.
- List the four most common bottlenecks on a system.

Module 2 — Performance Tools

- Identify various performance tools available on HP-UX.
- Categorize each tool as either real time or data collection.
- List the major features of the performance tools.
- Compare and contrast the differences between the tools

Module 3 — GlancePlus

- Compare GlancePlus with other performance monitoring/management tools.
- Start up the GlancePlus terminal interface (**glance**) and graphical user interface (**gpm**).

Overview

Module 4 — Process Management

- Describe the components of a process.
- Describe how a process executes, and identify its process states.
- Describe the CPU scheduler.
- Describe a context switch and the circumstances under which context switching occurs.
- Describe in general, the HP-UX priority queues.

Module 5 — CPU Management

- Describe the components of the processor module.
- Describe how the TLB and CPU cache are used.
- List four CPU related metrics.
- Identify how to monitor CPU activity.
- Discuss how best to use the performance tools to diagnose CPU problems.
- Specify appropriate corrections for CPU bottlenecks.

Module 6 — Memory Management

- Describe how the HP-UX operating system performs memory management.
- Describe the main performance issues that involve memory management.
- Describe the UNIX buffer cache.
- Describe the `sync` process.
- Identify the symptoms of a memory bottleneck.
- Identify global and process memory metrics.
- Use performance tools to diagnose memory problems.
- Specify appropriate corrections for memory bottlenecks.
- Describe the function of the `serialize` command.

Module 7 — Swap Space Performance

- Describe the difference between swap usage and swap reservation.
- Interpret the output of the **swapon** command.
- Define and configure pseudo swap.
- Define and configure swap space priorities.
- Define and configure **swchunk** and **maxswapchunks**.

Module 8 — Disk Performance Issues

- List three ways disk space can be used.
- List disk device files.
- Identify disk bottlenecks.
- Identify kernel system parameters.

Module 9 — File System Performance

- List three ways file systems are used.
- List basic file system data structures.
- Identify file system bottlenecks.
- Identify kernel system parameters.

Module 10 — VxFS Performance

- Understand JFS structure and version differences
- Explain how to enhance JFS performance
- Set block sizes to improve performance
- Set Intent-Log size and rules to improve performance
- Understand and manipulate synchronous and asynchronous IO
- Identify JFS tuning parameters
- Understand and control fragmentation issues
- Evaluate the overhead of online backup snapshots

Overview

Module 11 — NFS Performance

- List factors directly related to network performance.
- Describe how to determine network workloads (server and client).
- Evaluate UDP and TCP transport options.
- Identify a network bottleneck.
- List possible solutions for a network performance problem.

Module 12 — Tunable Kernel Parameters

- Identify which tunable parameters belong to which category
- Identify tunable kernel parameters that could impact performance
- Tune both static and dynamic tunable parameters

Module 13 — Putting It All Together

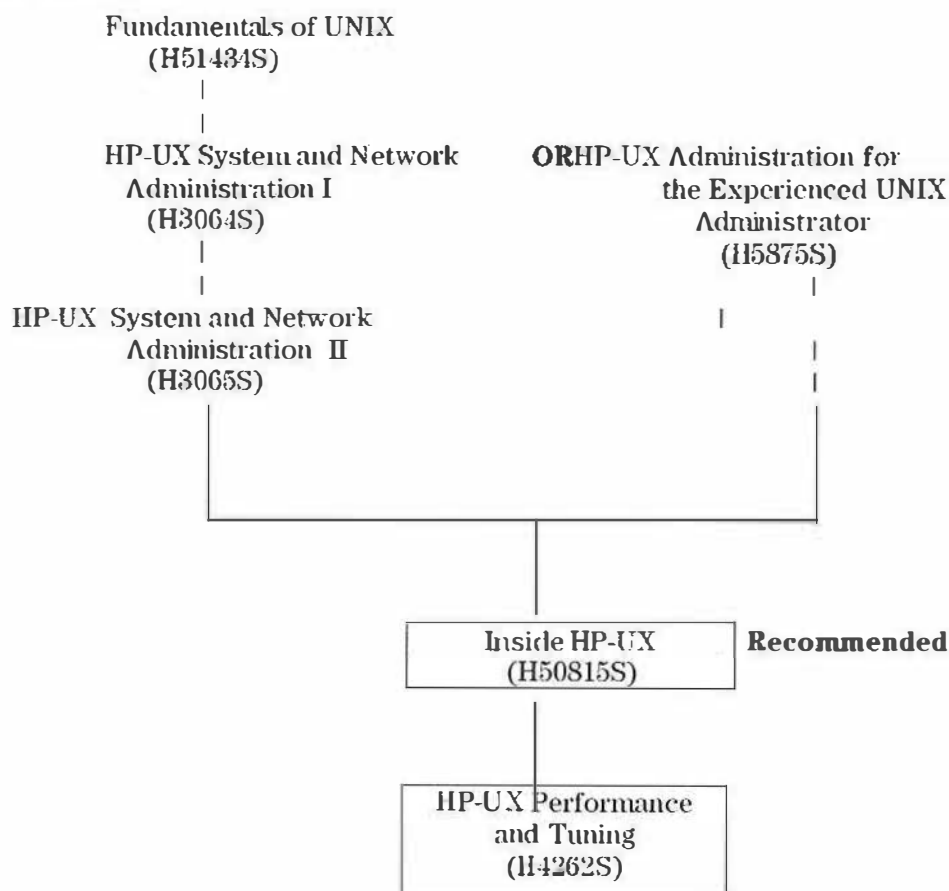
- Identify and characterize some network performance problems.
- List some useful tools for measuring network performance problems and state how they might be applied.
- Identify bottlenecks on other common system devices not associated directly with the CPU, disk, or memory.

Student Profile and Prerequisites

The student should be well versed in UNIX and able to perform the usual duties of a system administrator. Students should have completed *HP-UX System and Network Administration I* and *HP-UX System and Network Administration II* prior to attending this course or equivalent experience on another manufacturer's equipment.

NOTE: The course *Inside HP-UX (H5081S)* is not a formal prerequisite to attending *HP-UX Performance and Tuning*, but it should be considered a co-requisite training course for the serious HP-UX Performance Specialist. (The suggested order is *Inside HP-UX* then *HP-UX Performance Tuning*, but as the two courses have a synergistic relationship, the order is not absolute).

Curriculum Path



Overview

Agenda

The following agenda is only a guideline. The instructor may vary it if desired.

The course will run until the afternoon of the third day. The last hour or so can be used to demonstrate more fully the performance offerings, such as HP PRM and HP PerfView.

Day 1

- 1 — Introduction
- 2 — Performance Tools

Day 2

- 3 — GlancePlus
- 4 — Process Management
- 5 — CPU Management

Day 3

- 6 — Memory Management
- 7 — Swap Space Performance Managing
- 8 — Disk Performance Issues

Day 4

- 9 — File System Performance
- 10 — VxFS Performance
- 11 — NFS Performance

Day 5

- 12 — Tunable Kernel Parameters
- 13 — Putting It All Together

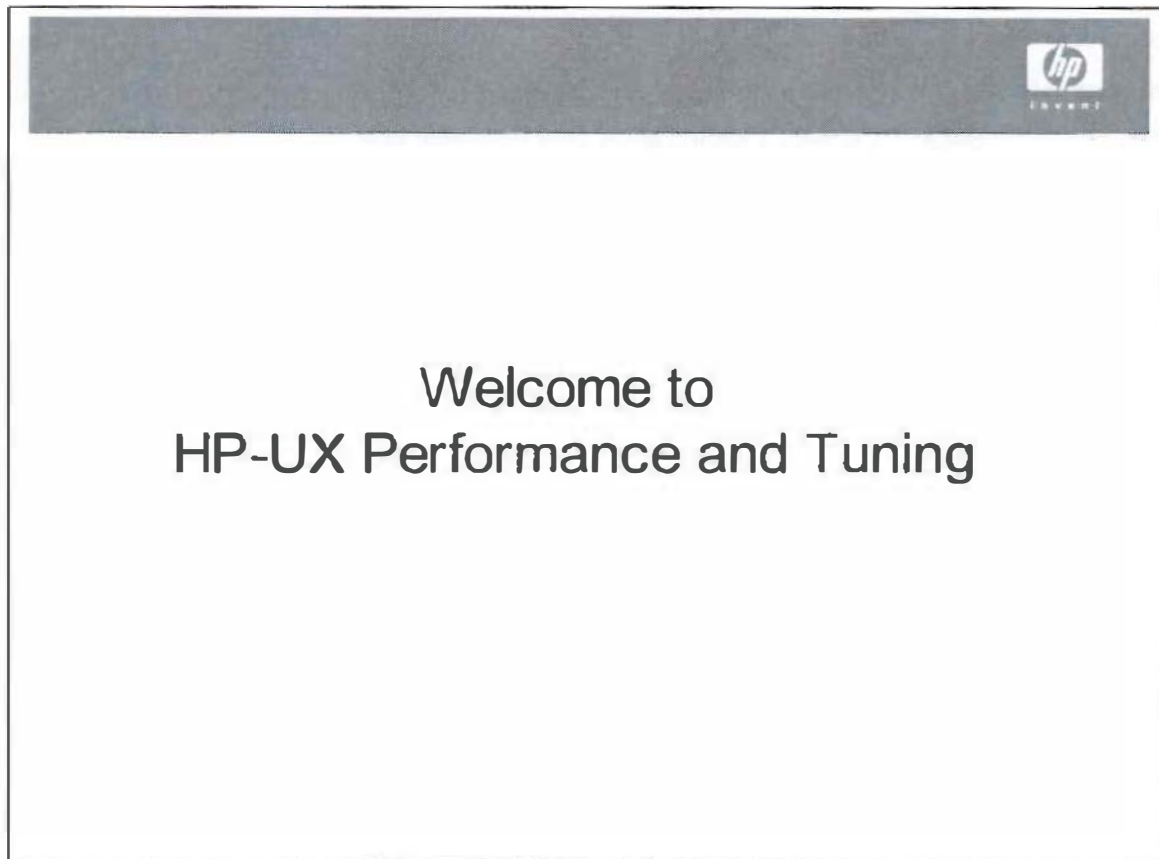
Module 1 — Introduction

Objectives

Upon completion of this module, you will be able to do the following:

- List characteristics of a system yielding good user response time.
- List characteristics of a system yielding high data throughput.
- List three generic areas most often analyzed for performance.
- List the four most common bottlenecks on a system.

1-1. SLIDE: Welcome to HP-UX Performance and Tuning



Student Notes

Welcome to the *HP-UX Performance and Tuning* course.

This course is designed to provide a high level understanding of common performance problems and common bottlenecks found on an HP-UX system.

The course uses HP performance tools to view activity currently on the system. While many tools can be used to analyze the activity, this course primarily utilizes the **glance** tool, which is specifically tailored for HP-UX systems.

1-2. SLIDE: Course Outline

Course Outline



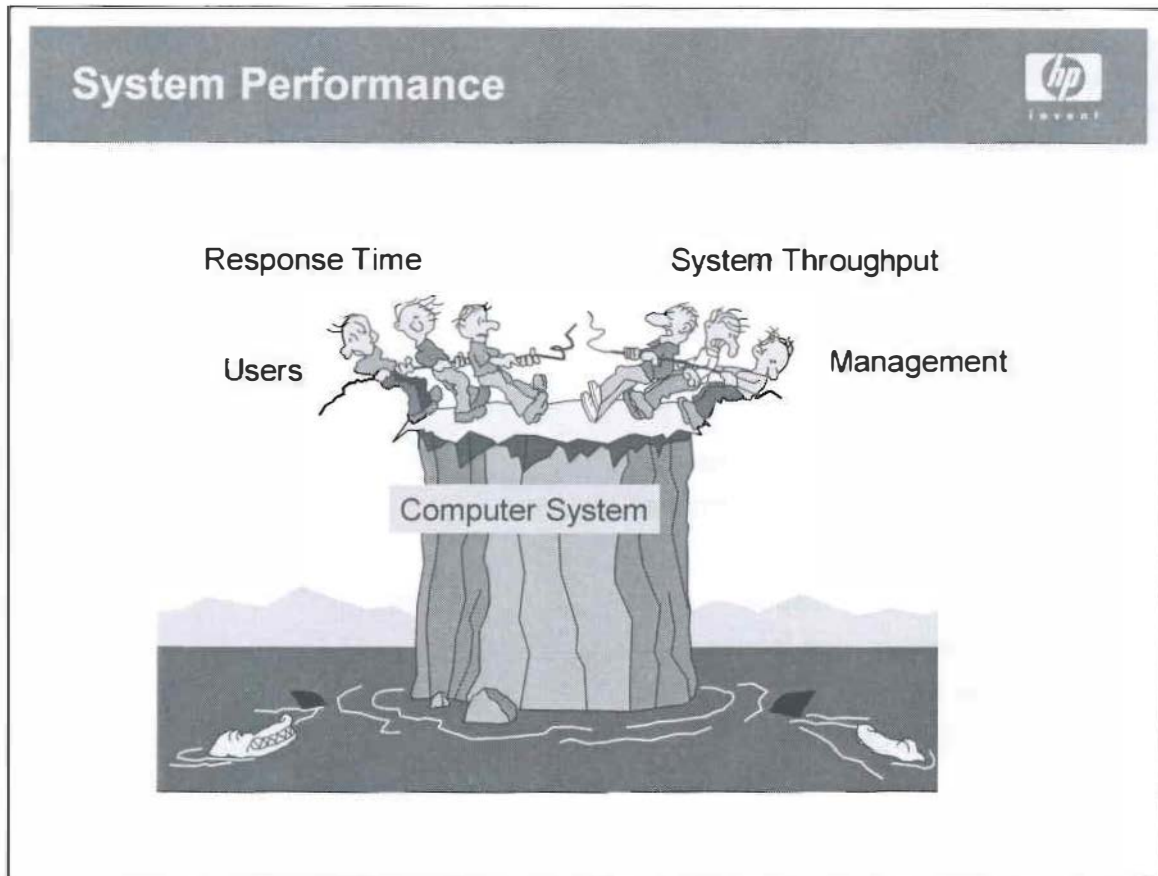
- Introduction to Performance
- Performance Tools – Overview
- GlancePlus
- Process Management
- CPU Management
- Memory Management
- Swap Space Performance Issues
- Disk and File System Performance Issues
- HFS Performance Issues
- VxFS Performance Issues
- Network Performance Issues
- Tuning the Kernel
- Putting it All Together – Performance Recap

Student Notes

Topics covered in this course include:

- **System Internals**
This module includes information related to how the system components (CPU, memory, file systems, and network) function and interact with each other. Similar to a mechanic not being able to tune a car's engine until he understands how it works, a system administrator cannot tune system resources properly until he has a good understanding of how the resources work.
- **Performance Tools**
There are many performance tools that are available with HP-UX. Some tools come as standard equipment; other tools are additional add-on products. Some tools provide runtime monitoring; other tools perform data collection. We will review all of the tools.
- **Specialty Areas**
These modules cover areas of special interest to customers in particular types of environments. Three specialty areas are covered at a high level. These are NFS and networking, databases, and application profiling.

1-3. SLIDE: System Performance



Student Notes

Different computer systems have different requirements. Some systems may need to provide quick response time; other systems may need to provide a high level of data throughput.

Response Time — User's Perspective

Response time is the time between the instant the user presses the return key or the mouse button and the receipt of a response from the computer. Users often use response time as a criterion of system performance.

A system that yields high response is typically *not* 100% utilized. Often there are free CPU cycles, along with low utilization of disk drives, and with no swapping or paging. Because the system resources are not being utilized constantly, often when a user executes a task the resources are available immediately, yielding quick response time to the user.

Users want low utilization of resources in order to experience optimal response time.

Throughput — IT or MIS Management Perspective

Throughput is the number of transactions accomplished in a fixed-time period. Management is often interested in how many compilations or how many reports they can generate in a specific amount of time. Many systems use benchmarks (like SPECmarks or TPC), which measure, in general, how many operations or transactions a system can perform per minute.

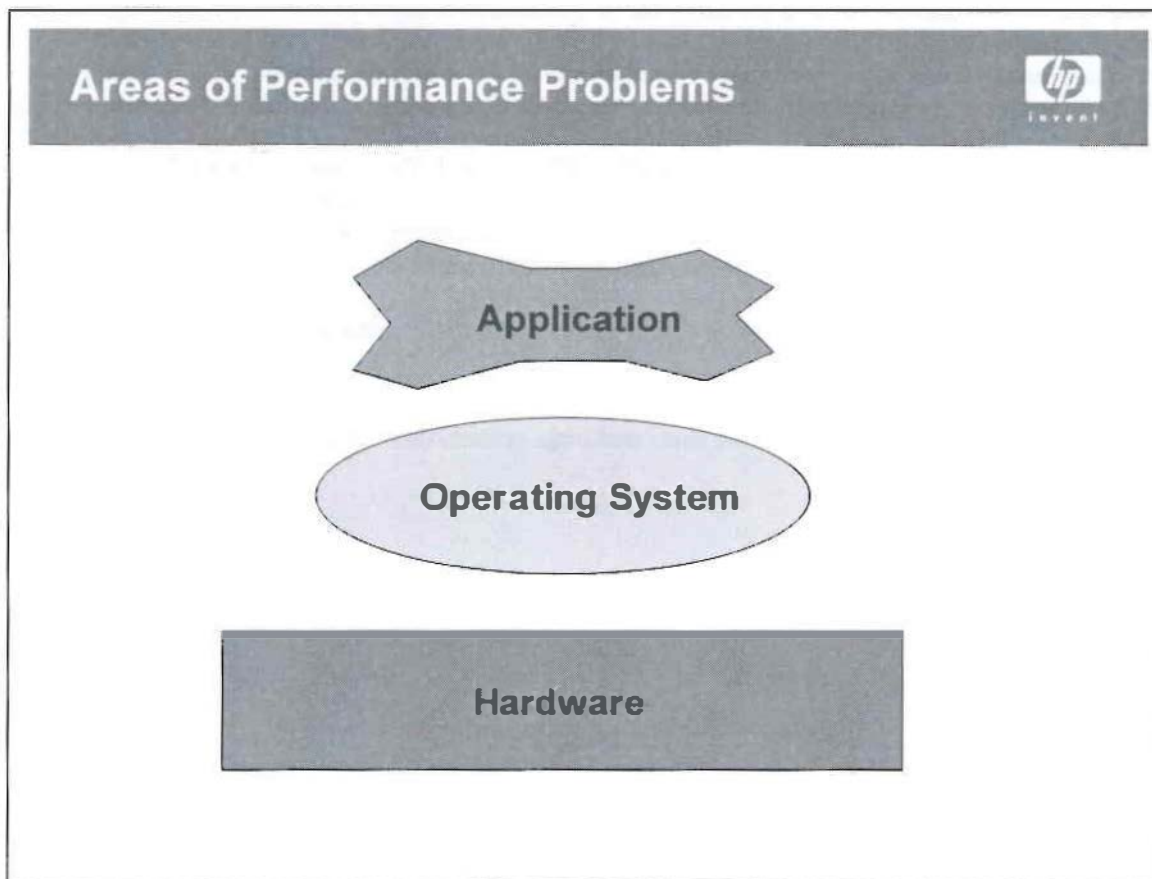
A system that yields high throughput is typically 100% utilized. There are no free CPU cycles; there are always jobs in the CPU run queue; the disk drives are constantly being utilized; and there is often pressure on memory. Because the system resources are constantly in use, the amount of work produced typically yields good system throughput.

Management wants high utilization of resources to maximize system performance.

Question

Is it possible to get both good response time and high system throughput?

1-4. SLIDE: Areas of Performance Problems



Student Notes

This slide shows a hierarchical view of a computer system. The base of a computer is its hardware. Built on top of the hardware is the operating system (i.e. the operating system is dependent on the hardware in order to run). The application programs are built on top of the operating system (OS).

All three of these areas can have performance problems.

Hardware

The hardware moves data within the computer system. If the hardware is slow, then, no matter how finely tuned the OS and applications are, the system will still be slow. Ultimately, the system is only as fast as the hardware can move the data.

Items affecting the speed of the hardware include CPU clock speed, amount and speed of memory, type of disk controller (Fast/Wide SCSI or Single-Ended SCSI), and type of network card (FDDI or Ethernet).

Operating System

The operating system runs on top of the hardware. It controls how the hardware is utilized. The operating system decides which process runs on the CPU, how much memory to allocate for the buffer cache, whether I/O to the disks is performed synchronously or asynchronously, and so on. If the operating system is not configured properly, then the performance of the system will be poor.

Items affecting how the operating system performs include process priorities and their nice values, the tunable OS parameters, the mount options used for file systems, and the configurations of network and swap devices

Applications

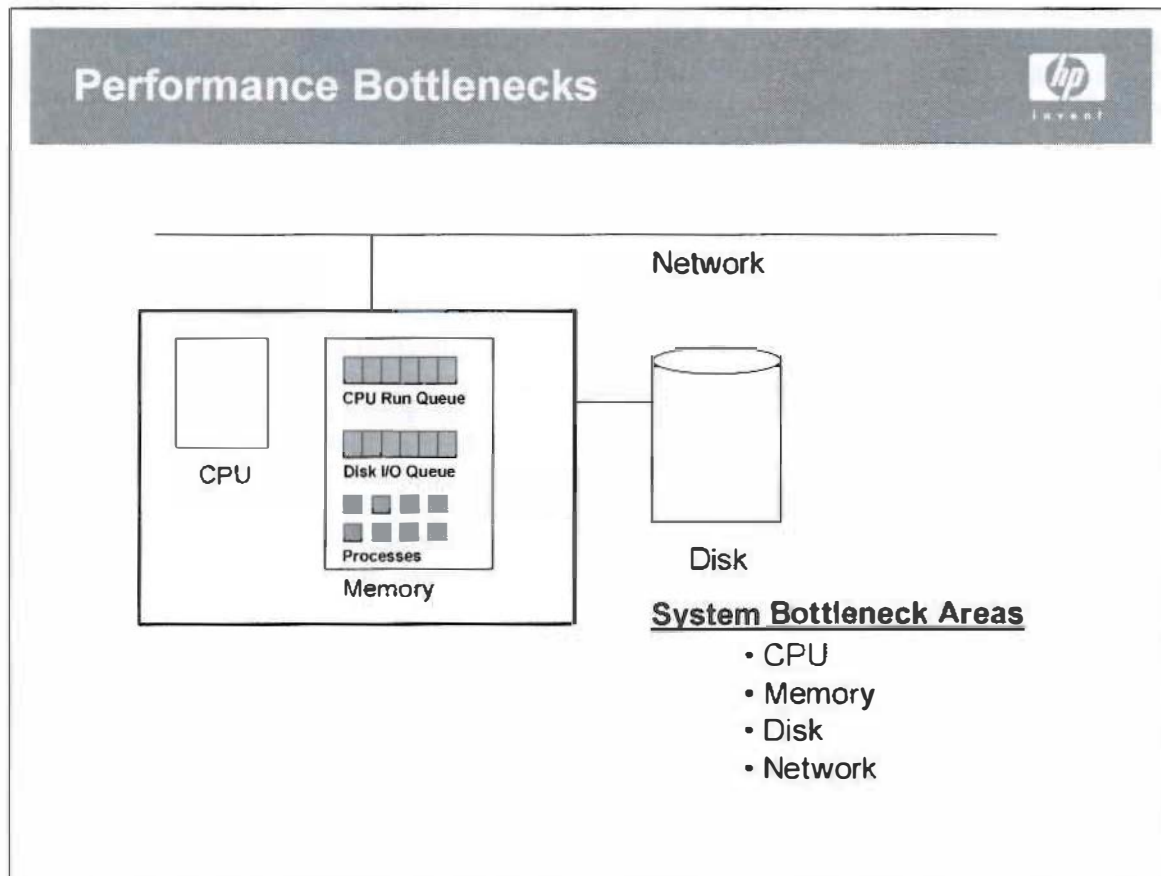
The applications run on top of the operating system. The application programs include software, such as database management systems, electronic design applications programs, and accounting-based applications. The performance of the application program is dependent on the operating system and hardware, but it is also dependent on how the application is coded, and how the application itself is configured.

Items affecting the performance of the application include how the application data is laid out on the disk, how many users are trying to use the application currently, and how efficiently the application uses the system's resources.

Questions

In which of these three areas are most performance problems located?

1-5. SLIDE: Performance Bottlenecks



Student Notes

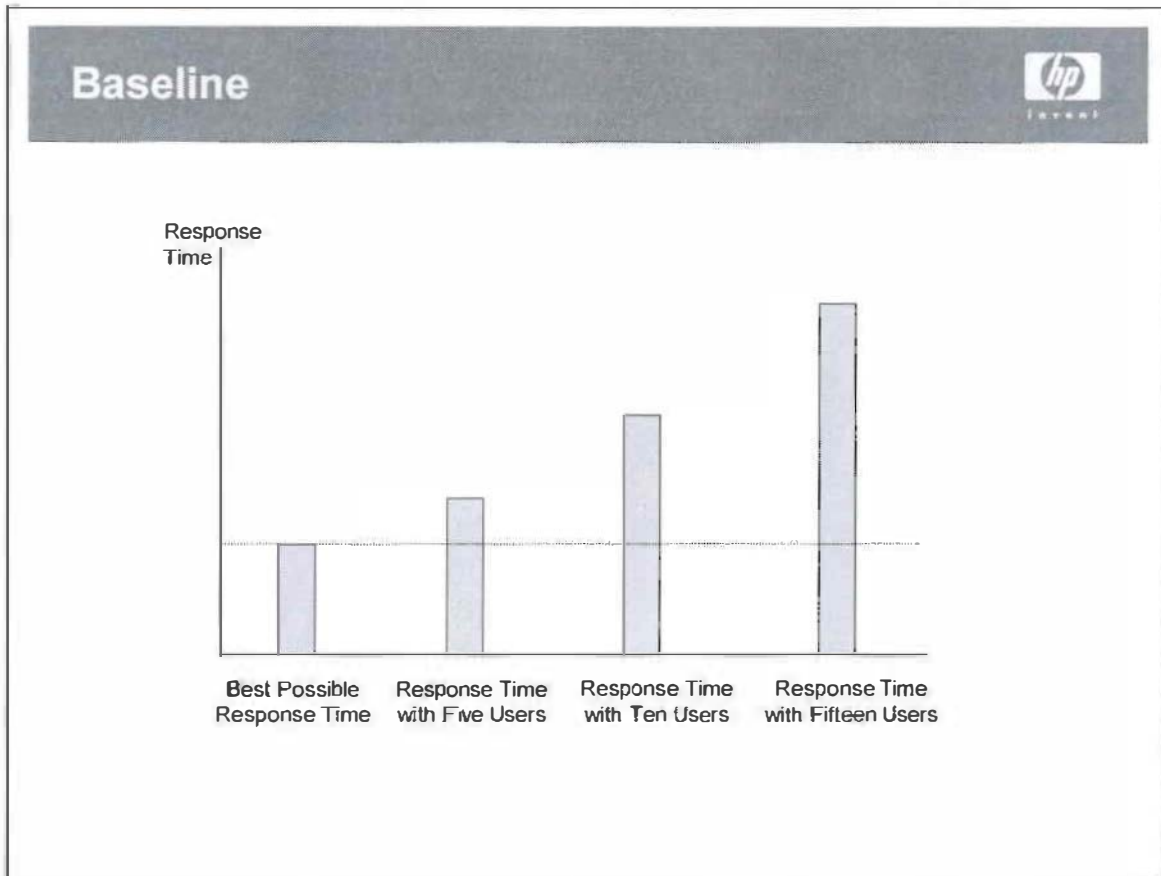
Poor performance often results because a given resource cannot handle the demand being placed upon it. When the demand for a resource exceeds the availability of the resource, then a **bottleneck** exists for that resource.

Common resource bottlenecks are:

- CPU** A CPU bottleneck occurs when the number of processes wanting to execute is constantly more than the CPU can handle. Basic symptoms of a CPU bottleneck are high CPU Utilization and multiple jobs in the CPU run queue, consistently.
- Memory** A memory bottleneck occurs when the total number of processes on the system will not all fit into memory at one time (i.e. there are more processes than memory can hold). When this happens, pages of memory need to be copied out to the swap partition on disk to free space in memory. Basic symptoms of a memory bottleneck are high memory utilization and consistent I/O activity to the swap partition on disk.

- Disk A disk bottleneck occurs when the amount of I/O to a specific disk is more than the disk can handle. Basic symptoms of a disk bottleneck include high utilization of a disk drive and multiple I/O requests consistently in the disk I/O queue.
- Network A network bottleneck occurs when the amount of time needed to perform network-based transactions is consistently greater than expected. Basic symptoms of a network bottleneck include network collisions, network request timeouts, and packet retransmissions.

1-6. SLIDE: Baseline



Student Notes

In order to quantify good versus poor performance, a customer needs to know what the best possible response time for a given workload can be. The procedure for calculating the best possible response time for a given workload is known as **baselining**.

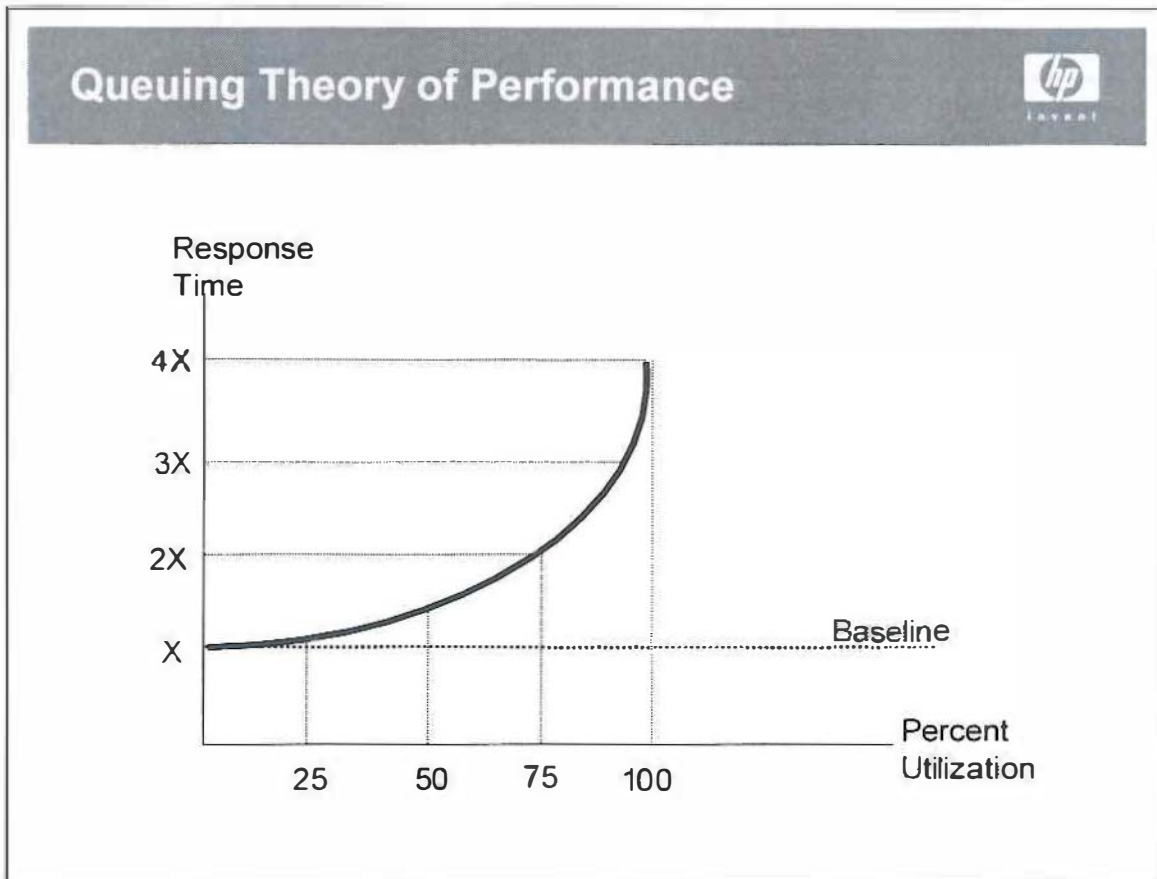
To calculate the baseline (i.e. the best possible response time) for a particular workload, the workload needs to be performed when no other activity is on the system. The intent is that when all resources are free, the workload will be able to execute as quickly as possible, thereby yielding the best possible response time.

Once the baseline value is known, a relative measure is now available for determining how poorly the workload is performing. For example, assume a baseline value of 5 seconds for the workload shown on the slide. When five users are on the system, the response time for the workload increases to 7 seconds. The relative comparison shows response time taking 40% (or 2 seconds) more time to perform this workload when five users are on the system. We have just quantified the relative effect of having five users on the system relative to this particular workload.

The slide illustrates the typical behavior for a given workload. As more users concurrently utilize the system, the response time for a given workload gets worse.

NOTE: In this class we will run baseline metrics using simplified "workload" simulation programs. Results will vary greatly with your applications.

1-7. SLIDE: Queuing Theory of Performance



Student Notes

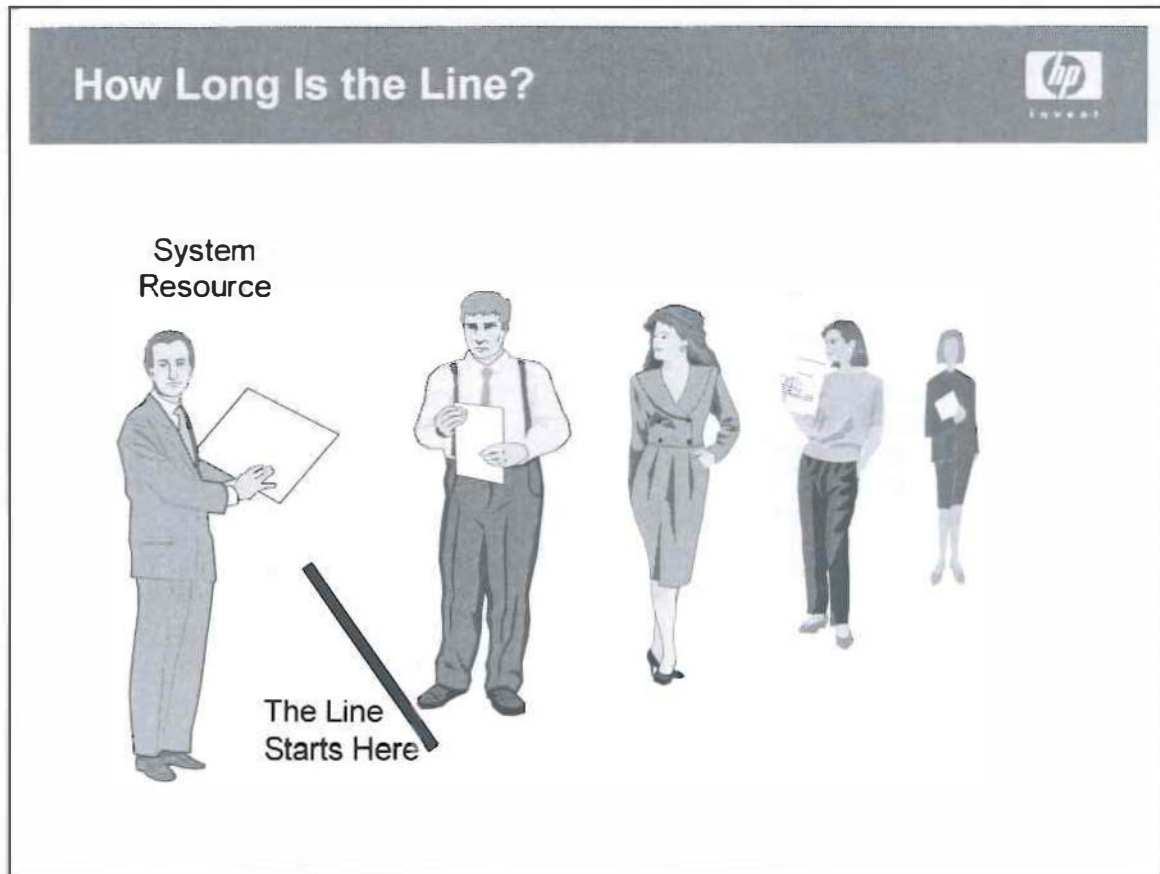
The queuing theory of performance states that the *average* response time of a given resource is directly linked to the *average* utilization of that resource.

The slide shows a baseline value of X seconds for a given resource. According to the queuing theory, the users will experience this response time when the resource has an average utilization of 0 to 25%. When the average utilization of the resource reaches 75%, the average response time will *double*. As the average utilization approaches 100%, the average response time *quadruples*.

The bottom line is, as the average utilization of the resource increases, the average response time gets worse and worse.

Why does the average response time become poor as the average utilization of a resource increases?

1-8. SLIDE: How Long Is the Line?



Student Notes

The reason why the average response time gets so poor when the average resource utilization increases, is because the length of the line waiting to get to the resource gets longer. As resource utilization increases, the number of jobs waiting on the resource also increases.

When poor performance is experienced, it is most often due to the length of the queue becoming long. A long queue causes jobs to spend most of their time waiting in line for the resource (CPU, memory, network, or disk), as opposed to being serviced by the resource.

The slide shows four people waiting in line to get to a resource (think of a line in a bank with one bank teller). If it takes 5 minutes to service one customer, then the fourth person in line will wait 15 minutes before reaching the resource. Adding another 5 minutes to service, the request brings the total response time to 20 minutes for the last person in line, as opposed to 5 minutes if the line had been empty. Of course there is also an overhead experienced because of "switching" from one customer to the next. This switching is minimal in this example because the customers are handled in a serial fashion.

1-9. SLIDE: Example of Queuing Theory

Example of Queuing Theory							
sar -d 5 5							
utilization of disk							
15:31:55	device	%busy	avque	r+w/s	blks/s	avwait	avserv
15:32:00	c0t6d0	81	3.4	31	248	59.31	21.20
	c0t5d0	5	.5	1	32	0.65	23.58
15:32:05	c0t6d0	84	3.5	34	245	71.64	24.04
	c0t5d0	3	.5	2	8	0.25	17.93
15:32:10	c0t6d0	68	2.9	31	248	51.36	18.55
	c0t5d0	1	.5	0	6	0.48	19.18
15:32:15	c0t6d0	71	2.7	30	30	62.88	24.16
	c0t5d0	0	.5	1	3	0.65	29.25
15:32:20	c0t6d0	69	2.7	29	29	61.70	24.14
	c0t5d0	0	.5	1	3	0.65	29.25

Student Notes

The above slide provides an example of the queuing theory for the disk drives as reported with the **sar** tool. The four fields to focus on are:

- %busy** The percentage of utilization of each disk
- avque** The average number of I/O requests in the queue for that disk
- avwait** The average amount of time a request spends waiting in that disk's queue
- avserv** The average amount of time to service that I/O request (not including the wait time)

Analyzing the data shows a baseline around 20 milliseconds to service an I/O request (approximate average of **avserv** column).

The first line item shows a disk that is 81% utilized. The total response time is the **average wait** plus the **average service**, or approximately 80 milliseconds. This is four times longer than the baseline of 20 milliseconds. In fact, each snapshot shows the busy disk waiting in the queue for an amount of time greater than the amount of time to service the I/O request. To

see why the wait time is so high, look at the **avque** size. *Notice the queue size is highest when the device is most busy.* This is the basic concept of the performance queuing theory.

1-10. SLIDE: Summary

Summary



- Objective for the system:
 - Provide fast response time to users, or
 - Maximize throughput of system
- Three performance problem areas:
 - Hardware
 - Operating System
 - Application
- Performance bottlenecks:
 - CPU
 - Disk
 - Memory
 - Network
- Need for baselines
- Performance queuing theory

Student Notes

To summarize this module, systems are tuned for *response time* or for *throughput*. This class focuses on tuning for best possible response time.

Areas that affect response time are speed of the hardware, configuration of the operating system, and configuration of the application. This class focuses on the configuration of the operating system.

Common bottlenecks with computer systems include CPU, memory, disk, and network. This class discusses all four bottlenecks.

Baselines are an important measurement tool for quantifying performance. In the lab for this module, the student will establish CPU and disk I/O baselines.

Finally, the queuing theory of performance states that the average response time increases as the average utilization of a resource increases. This is an important concept, which will be revisited throughout this course.

1-11. LAB: Establishing a Baseline

Directions

The following lab exercise establishes baselines for three CPU-bound applications and one disk-bound application. The objective is to time how long these applications take when there is no activity on the system.

These same applications will be executed later on in the course when other bottleneck activity is present. The impact of these bottlenecks on user response time will be measured through these applications.

1. Change directory to `/home/h4262/baseline`.

```
# cd /home/h4262/baseline
```

2. Compile three C programs **long**, **med** and **short** by running the `BUILD` script

```
# ./BUILD
```

3. Time the execution of the **long** program. Make sure there is no activity on the system.

```
# timex ./long
```

```
Record Execution Time  real:
                        user:
                        sys:
```

4. Time the execution of the **med** program. Make sure there is no activity on the system.

```
# timex ./med
```

Cpu = user + sys = real

```
Record Execution Time  real:
                        user:
                        sys:
```

5. Time the execution of the **short** program. Make sure there is no activity on the system.

```
# timex ./short
```

```
Record Execution Time  real:
                        user:
                        sys:
```

6. Time the execution of the **diskread** program.

```
# timex ./diskread
```

as user + sys # real

```
Record Execution Time  real:
                        user:
                        sys:
```

Module 1
Introduction

7. In the case of the long, med, and short programs the real time is the sum of the `usr` and `sys` time (approximately). This is not the case with `diskread`. Explain why.

1-12. LAB: Verifying the Performance Queuing Theory

Directions

The performance queuing theory states that as the number of jobs in a queue increases, so will the response time of the jobs waiting to use that resource.

This lab uses the **short** program compiled from
`/home/h4262/baseline/prime_short.c`.

1. In terminal window 1, monitor the CPU queue with the **sar** command.

```
# sar -q 5 200
```

2. In a second terminal window, time how long it takes for the **short** program to execute.

```
# timex ./short &
```

How long did the program take to execute? 9.42

How does this compare to the baseline measurement from earlier? _____

What is the CPU queue size? _____

3. Time how long it takes for three **short** programs to execute.

```
# timex ./short & timex ./short & timex ./short &
```

How long did the slowest program take to execute? 28

How did the CPU queue size change from step 2? 5

4. Time how long it takes for five **short** programs to execute.

```
# timex ./short & timex ./short & timex ./short & \
  timex ./short & timex ./short &
```

How long did the slowest program take to execute? 56

How did the CPU queue size change from step 3? 8

5. Is the relationship between elapsed execution (real) time and the number of running programs linear?
6. Comment about the overhead of switching from one process to another.

Module 1
Introduction

Module 2 — Performance Tools

Objectives

Upon completion of this module, you will be able to do the following:

- Identify various performance tools available on HP-UX.
- Categorize each tool as either real time or data collection.
- List the major features of the performance tools.
- Compare and contrast the differences between the tools.

2-1. SLIDE: HP-UX Performance Tools



Student Notes

Many performance tools are available for many different purposes. In the HP-UX operating system, there are over 50 different performance-related tools.

Some tools provide real-time performance information, such as, "How busy the CPU is right now?" Other tools collect data in the background and maintain a history of performance information. This module addresses all the tools and the different functions they perform.

2-2. SLIDE: HP-UX Performance Tools (Continued)

HP-UX Performance Tools



Objective:

- Identify the various performance tools available on HP-UX
- Demonstrate their mechanics
- Discuss their features
- Compare and contrast the differences between the tools

Student Notes

The objective of this module is to highlight *all* the performance tools available with HP-UX, to categorize them by function, and to describe how each tool is used.

The module is intended to be a quick reference of performance tools, which the student can refer to when needing to select a tool for a specific task.

NOTE: This module does *not* discuss how to interpret the output of the tools. Interpretation of the metrics is provided in later modules.

2-3. SLIDE: Sources of Tools

Sources of Tools



- Standard tools
 - Tools found on UNIX systems, including HP-UX
 - Tools frequently found on other UNIX systems
- HP-UX-specific tools
 - Tools found only on HP-UX
- Optional tools
 - Tools licensed and sold separately
(Generally available only on HP-UX)

Student Notes

Three types of tools are presented in this module:

- **Standard Tools**
Standard tools are those frequently found on many UNIX systems, including HP-UX. The advantage of the standard tools is that their results can be compared with those being collected on other UNIX platforms. This provides an "apples for apples" comparison, which is desirable when comparing systems.

The output from these standard tools (and some of the options) may vary slightly among UNIX systems. In addition, differences between the various UNIX implementations can affect the reliability of the metrics being output by the tools. Therefore, be careful to check the results with other tools or seek help before basing important tuning decisions on the value of one metric.

- **HP-Specific Tools**
HP-specific tools are those which are found only on HP-UX operating systems. These tools are often tailored specifically to understand HP-UX implementations.

These tools are generally *not* found on other UNIX implementations, as other

implementations are different from those of HP. Some of the HP-specific tools come with the base OS; others are purchased as optional tools.

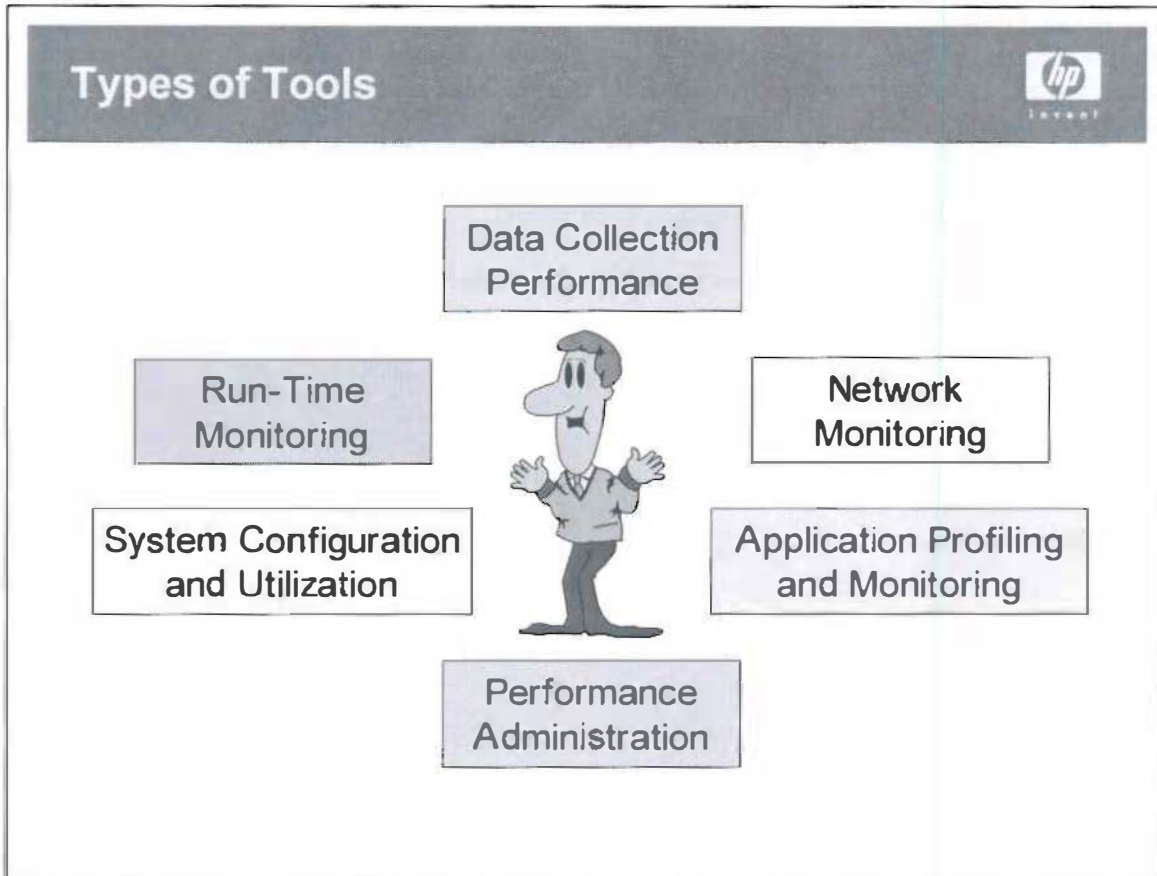
- **Optional Tools**

Optional tools are tools that are added to the operating system in addition to the standard tools. Some of the optional tools, such as the HP-PAK (Programmers Analysis Kit), may be included with add-on software, such as compilers for HP-UX. Other optional tools, like GlancePlus, PerfView, MeasureWare, NetMetrix, PRM (Process Resource Manager), and WLM (Work Load Manager), are purchased individually or in small bundles (GlancePlus Pak also includes a MeasureWare agent).

Optional tools are typically licensed from HP. They offer many advantages over the standard tools including:

- ease of use
- accuracy
- granularity
- low overhead
- additional metrics

2-4. SLIDE: Types of Tools



Student Notes

The tools covered in this section fall into six main categories:

- **Run-Time Monitoring Tools**
These tools provide information as to the performance of the system *now*. The information is current and provides a real-time perspective as to the state of the system at the current moment.
- **Data Collection Performance Tools**
These tools collect performance data in the background, summarize or average the data into a summary record, and log the summary record to a file or files on disk. They do not typically provide real-time data.
- **Network Monitoring Tools**
These tools monitor performance, status, and packet errors on the network. They include both monitoring and configuration tools related to network management.
- **Performance Administrative Tools**
A system administrator can use these tools to manage the performance of his system.

They typically do not *report* any data, but allow the current configuration of the system (and its components) to be *changed* to help improve performance

- **System Configuration and Utilization Information Tools**

These tools *report* current system configurations (such as LVM and file systems). They also report utilization of resource statistics, like disk and file system space and number of processes.

- **Application Profiling and Monitoring Tools**

These tools provide in-depth analysis about the behavior of a program. These tools monitor and trace the execution of a process, and report the resources and calls made during its execution.

2-5. SLIDE: Criteria for Comparing the Tools

Criteria for Comparing the Tools



- Source of data
- Scope
- Additional cost versus no cost
- Intrusiveness
- Accuracy
- Ease of use
- Portability
- Metrics available
- Data collection and storage
- Permissions required

Student Notes

Each tool has strengths and weaknesses, advantages and disadvantages, and unique features. Some items to consider when selecting a tool are:

Source of Data	The collected data can come from a variety of sources, including the kernel, an application, or a specific daemon (like the <code>midaemon</code>).
Scope	The scope determines the level of detail provided with the tool. Most of the standard tools do not show process-level metrics. For example, they display global disk I/O rates, but do not show which process is generating the I/O or the disk on which the I/O is concentrated.
Cost	The cost sometimes determines if the tool is an option. Many of the HP-specific tools have additional costs associated with them. (Many of these tools have evaluation copies available for a trial period.)
Intrusiveness	The intrusiveness relates to the overhead associated with running the tool. Some tools also have significant overhead. A large user community using <code>top</code> , for example, may be responsible for generating large amounts of "monitoring" overhead on the system. Another example is the <code>ps</code>

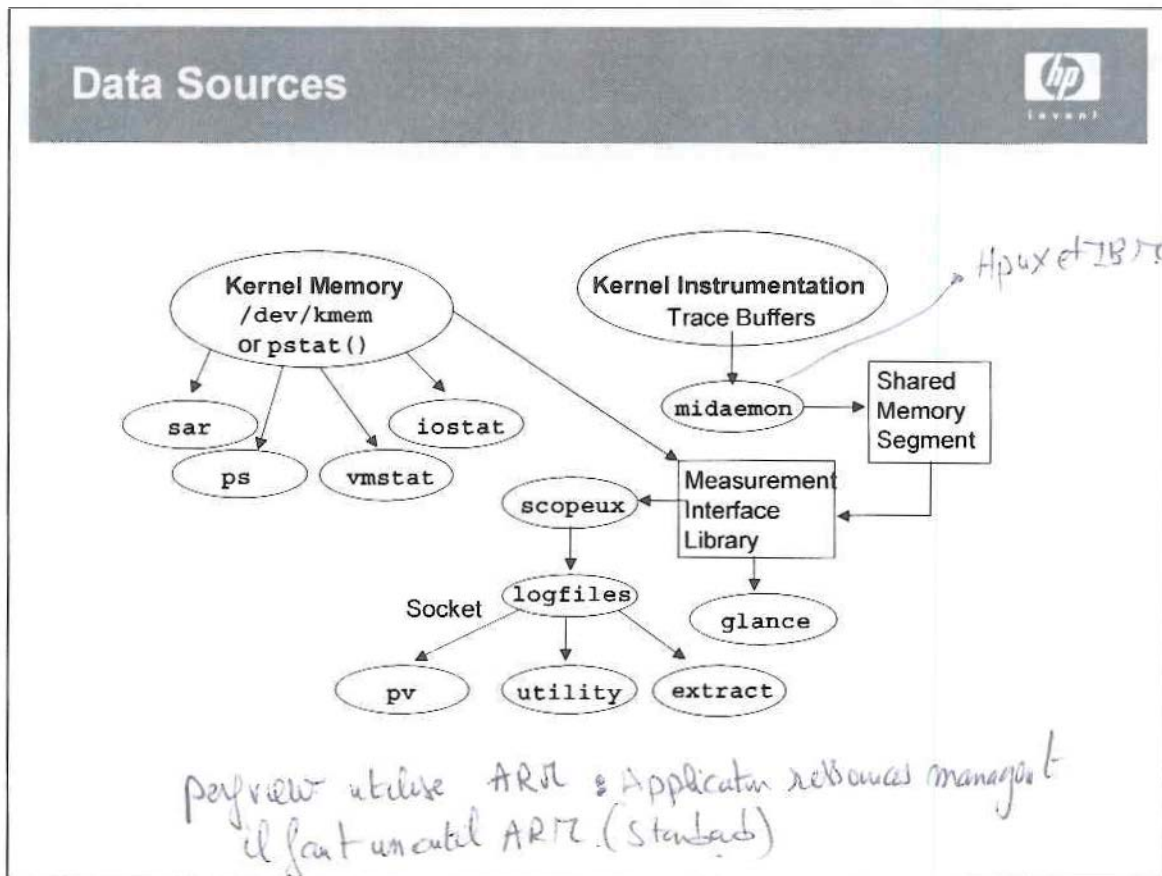
command. It has little impact on most systems due to the low frequency at which it is executed. However, the `ps` command places fairly high overhead on the system during its execution.

Accuracy The accuracy of the tool relates to the reliability of the data being reported. Many standard UNIX tools, like `vmstat` and `sar`, have been ported from other UNIX systems. The registers that they monitor may not always correspond to the registers that the kernel updates.

Others There are other factors that can have significant impact on the tool you decide to use. These factors include familiarity, metrics available, permissions required, and portability.

As the tools are presented in the upcoming pages, many of these items will be addressed.

2-6. SLIDE: Data Sources



Student Notes

The standard tools read information from the UNIX counters and registers maintained in kernel memory (accessible via the `/dev/kmem` device file and the `pstat()` system call). These counters and registers are updated 10 times a second as a standard part of most UNIX system implementations. The data in the counters and registers are generally adequate for most performance jobs, but do not provide enough detail when in-depth tuning is needed.

The optional tools for HP-UX use an additional source called kernel instrumentation (KI). The KI interface provides additional information beyond the UNIX kernel counters and registers. The KI interface gathers performance information on a *system call* basis, with every system call generated by every process being traced. The KI interface uses a proprietary measurement interface library to derive the additional metrics. These tools are frequently revised and updated to provide the highest levels of accuracy with the lowest possible overhead. The optional tools, such as Glance and MeasureWare, are KI-based tools when running on HP-UX systems, although they are available for other vendor systems as well.

Additional information about KI-based tools (also known as resource and performance management (RPM) tools) can be obtained from the RPM web site at: www.hp.com/go/rpm

/var/cpt/perf/datafiles

2-7. SLIDE: Performance Monitoring Tools (Standard UNIX)

Performance Monitoring Tools (Standard UNIX)			
	Global Metrics	Process Details	Alarming Capability
<code>iostat</code>	Yes	No	No
<code>ps</code>	No	Yes	No
<code>sar</code>	Yes	No	No
<code>time</code>	No	Some	No
<code>timex</code>	Some	Some	No
<code>top</code>	Yes	Some	No
<code>uptime,w</code>	Some	Some	No
<code>vmstat</code>	Yes	No	No

Student Notes

The slide shows *run-time* performance monitoring tools included with HP-UX. These tools provide current information about the performance of the system. These tools are standard UNIX performance tools, which are found on most other UNIX implementations.

The **Global Metrics** column indicates whether the tool will show aggregate resource utilization without differentiating between specific resources.

The **Process Detail** column indicates whether the tool will show resources being used by a single PID.

The **Alarming Capability** column indicates whether the tool is capable of sending an alarm when one of the metrics exceeds a user-defined threshold.

2-8. TEXT PAGE: iostat

The `iostat` command reports I/O statistics for each active disk on the system.

Tool Source:	Standard UNIX (BSD 4.x)
Documentation	man page
Interval	>= 1 second
Data Source:	Kernel registers/counters
Type of Data:	Global
Metrics:	Physical Disk I/O
Logging:	Standard output device
Overhead:	Varies, depending on the output interval
Unique Features:	Terminal I/O
Full Pathname:	/usr/bin/iostat
Pros and Cons:	+ statistics by physical disk drive - limited statistics - poorly documented and cryptic headings

Syntax

```
iostat [-t] [interval [count]]
```

-t	Report terminal statistics as well as disk statistics
interval	Display successive lines summaries at this frequency
count	Repeat the summaries this number of times

Key Metrics

The `iostat` metrics include:

bps	Blocks (kilobytes) transferred per second
sps	Number of seeks per second
msps	Average milliseconds per seek

With the advent of new disk technologies, such as data striping, where a single data transfer is spread across several disks, the average milliseconds per seek becomes impossible to compute accurately. At best it is only an approximation, varying greatly, based on several dynamic system conditions. For this reason and to maintain backward compatibility, the milliseconds per seek (msps) field is set to the value 1.0.

Examples

```
# iostat 5 2
  device      bps      sps      msp
  c0t6d0         0       0.0       1.0
  c0t6d0      1100     34.6       1.0
```

```
# iostat -t 5 1

          tty          cpu
        tin tout        us  ni  sy  id
          0    0          2   0   1  98

device    bps      sps      msp
c0t6d0      0      0.0      1.0
```


2-9. TEXT PAGE: ps

The **ps** command displays information about selected processes running on the system. The command has many options for reducing the amount of output.

Tool Source:	Standard UNIX (BSD 4.x)
Documentation:	man page
Interval:	on demand
Data Source:	in core process table
Type of Data:	per process
Metrics:	state, priority, nice values, PIDs, times, ...
Logging:	Standard output device
Overhead	Varies, depending on the number of processes
Unique Features:	Wait channel and Run queue of processes.
Full Pathname:	/usr/bin/ps
Pros and Cons:	+ familiarity + options for altering output - minimal information - no averaging or summarization (i.e. no global metrics)

Syntax

```
ps [-aAcdefHjlp] [-C cmdlist] [-g grplist] [-G gidlist] [-n namelist]
    [-o format] [-R prmgrplist] [-s sidlist] [-t termlist]
    [-u uidlist] [-U uidlist]
```

Key Metrics

The **ps** metrics include:

ADDR	The memory address of the process, if resident; otherwise, the disk address.
C	Recent processor utilization, used for CPU scheduling (0-255).
F	Flags associated with the process (octal, additive): 0 Process is on the swap device 1 Process is in core memory 2 Process is a system process 4 Process is locked in memory (and many more)
NI	The nice value for the process; used in priority computation.
PPID	The process ID number of the parent process.
PID	The process ID number of this process.

PRI	The priority of the process.
S	The state of the process
	- I Process is being created (very rarely seen) - S Process is Sleeping - R Process is currently Runnable - T Process is Stopped (rare) - Z Process is terminated (aka Zombie process)
STIME	Starting time of process.
SZ	The size in 4-KB memory pages.
TIME	The cumulative execution time of the process.
TTY	The controlling terminal for the process.
WCHAN	The address of a structure representing the event or resource for which the process is waiting or sleeping.

Example

```
# ps -fu daemon
      UID    PID  PPID  C   STIME TTY          TIME COMMAND
  daemon  1171   1170   0  13:03:42 ?           3:10 /usr/bin/X11/X :0
  daemon  1565   1171   0  17:47:47 ?           0:00 pexd /tmp/te_pexd_1171.2 /dev/tty2
```

```
# ps -lu daemon
 F S      UID    PID  PPID  C PRI NI     ADDR   SZ    WCHAN TTY          TIME CMD
 1 S          1   1171   1170   1  154  20    dba00  697   3ace9c ?           3:10 X
 1 S          1   1565   1171   0  154  20    10e6900 115   3ace9c ?           0:00 pexd
```

2-10. TEXT PAGE: sar

The **sar** command collects and reports on many different system activities (and system areas), including CPU, buffer cache, disk, and others.

Related commands include **sadc**, **sa1**, and **sa2**. These commands are related to the data collection functionality of **sar** and will be addressed with the data collection commands.

Tool Source:	Standard UNIX (System V)
Documentation:	man page and kernel source
Interval:	>= 1 second
Data Source:	/dev/kmem registers/counters
Type of Data:	Global
Metrics:	CPU, Disk, and Kernel resources
Logging	Standard output device, or file on disk
Overhead:	Varies, depending on the output interval
Unique Features:	Disk I/O wait time, kernel table overflows, buffer cache hit ratios
Full Pathname:	/usr/sbin/sar
Pros and Cons:	+ familiarity + performs both real time and data collection functions - no per process information - no paging information, only designed for swapping (no longer done on HP-UX)

Syntax

```
sar [-ubdycwvmpAMSP] [-o file] t [n]
```

Metric-related options:

-u	CPU Utilization
-q	Run queue and swap queue lengths and utilization
-b	Buffer cache stats →
-d	Disk utilization
-y	TTY utilization
-c	System call rates
-w	Swap activity
-v	Kernel table utilization
-m	Semaphore and message queue utilization
-a	File access system routine utilization
-A	Everything!
-M	Per processor breakdown (used with -u and/or -q)
-P/p	Per processor set breakdown (used with -MU and/or -Mq)

Key Metrics

The **sar** command has many metrics. Included below are some sample metrics based on the disk and CPU reports:

CPU Report (-u)

The CPU report displays utilization of CPU and the percentage of time spent within the different modes.

%usr Percentage of time system spent in user mode
 %sys Percentage of time system spent in system mode
 %wio Percentage of time processes were waiting for (disk) I/O
 %idle Percentage of time system was idle

Disk Report (-d)

The disk report displays activity on each block device (i.e. disk drive).

Device Logical name of the device (device file name)
 %busy Percentage of time the device was busy servicing a request
 avque Average number of I/O requests pending for the device
 r+w/s Number of I/O requests per second (includes reads and writes)
 blks/s Number of 512-byte blocks transferred (to and from) per second
 await The average amount of time the I/O requests wait in the queue before being serviced
 avserv The average amount of time spent servicing an I/O request (includes seek, rotational latency, and data transfer times)

Examples

note de 500
 # sar -u 5 4 *CPU*
 HP-UX r3w14 B.10.20 C 9000/712 10/14/97

	%usr	%sys	%wio	%idle
08:32:24				
08:32:29	64	36	0	0
08:32:34	61	39	0	0
08:32:39	61	39	0	0
08:32:44	61	39	0	0
Average	61	39	0	0

Disk
 # sar -d 5 4
 HP-UX r3w14 B.10.20 C 9000/712 10/14/97

	device	%busy	avque	r+w/s	blks/s	await	avserv
08:32:24							
08:32:29	c0t6d0	19.36	0.55	20	1341	6.37	14.27
08:32:34	c0t6d0	26.40	0.58	27	1687	7.10	15.00
08:32:39	c0t6d0	21.00	0.54	23	1528	5.48	14.09
08:32:44	c0t6d0	21.00	0.54	23	1528	5.48	14.09
Average	c0t6d0	22.44	0.56	23	1552	6.34	14.45

2-11. TEXT PAGE: time, timex

Description

The **time** and **timex** commands report the elapsed (wall clock) time, the time spent in system mode, and the time spent in user mode, for a specific invocation of a program.

The **timex** command is an enhanced version of **time**, and can report additional statistics related to resources used during the execution of the command.

Tool Source:	Standard UNIX (System V)
Documentation:	man page and kernel source
Interval:	Process completion
Data Source:	Kernel registers/counters
Type of Data:	Process
Metrics:	CPU (user, system, elapsed)
Logging:	Standard output device
Overhead:	Minimal
Unique Feature:	Timing how long a process executes
Full Pathname:	/usr/bin/timex
Pros and Cons:	+ minimal overhead - cannot be used on already running processes

Syntax

`time command`

`timex [-o] [-p|f|b|k|m|r|t|] [-s] command`

- o List amount of I/O performed by command (requires **pacct** file to be present)
- s List activity (SAR data) present during execution of command (requires **sar** file to be present)

Example

```
timex find / 2>&1 >/dev/null | tee -a perf.data
```

```
real    39.49
user    1.47
sys     11.24
```

2-12. TEXT PAGE: top

Description

The **top** command displays a real-time list of the CPU consumers (processes) on the system, sorted, with the greatest users at the top of the list.

Tool Source:	Standard UNIX (BSD 4.x)
Documentation:	man page
Interval:	>= 1 second
Data Source:	Kernel registers/counters
Type of Data:	Global, Process
Metrics:	CPU, Memory
Logging:	Standard output device
Overhead:	Varies, depending on presentation interval
Unique Feature:	Real-time list of top CPU consumers
Full Pathname:	/usr/bin/top
Pros and Cons:	+ quick look at global and process CPU data - limited statistics - uses curses for terminal output

Syntax

```
top [-s time] [-d count] [-n number] [-q]

-s time    Set the delay between screen updates
-d count   Set the number of screen updates to "count", then exit
-n number  Set the number of processes to be displayed
-q         Run quick. The top command with a nice value of zero.
```

Key Metrics

The **top** metrics include:

SIZE	Total size of the process in KB. This includes text, data, and stack.
RES	Resident size of the process in KB. This includes text, data, and stack.
%WCPU	Average (weighted) CPU usage since top started.
%CPU	Current CPU usage over the current interval.

Example

```
* Start top with a 10 second update interval
# top -s 10

* Start top and display only 5 screen updates then exit
# top -d 5

* Start top and display only top 15 processes
# top -n 15
```

Module 2

Performance Tools

```
* Start top and let it run continuously
# top
System: r3wl4  Fri Oct 17 10:24:23 1997
Load averages: 0.55, 0.37, 0.25
115 processes: 113 sleeping, 2 running
Cpu states:LOAD  USER   NICE    SYS  IDLE  BLOCK  SWAIT  INTR  SSYS
              0.55   9.9%   0.0%   2.0% 88.1%  0.0%   0.0%  0.0%  0.0%

Memory: 24204K (15084K) real, 46308K (33432K) virtual, 2264K free  Page# 1/9
TTY  PID USERNAME PRI NI  SIZE  RES STATE  TIME %WCPU %CPU COMMAND
?    680 root      154 20  1328K  468K sleep 33:23 12.36 12.34 snmpdm
?    728 root      154 20   340K  136K sleep 18:20  5.82  5.81 mib2agt
?   1141 root      154 20 12784K 3708K sleep 84:06  4.47  4.47 netmon
?   1071 root       80 20  1264K  568K run   0:19  3.00  2.99 pmd
?   3892 root      179 20   308K  296K run   0:00  2.59  0.34 top

* To go to the next/previous page, type "j" and "k" respectively
* To go to the first page, type "t"
```

NOTE: The two values preceding real and virtual memory are the memory allocated for all processes, and in parentheses, memory allocated for processes that are currently runnable or that have executed within the last 20 seconds.

NOTE: **swait** and **block** are relevant for SMP systems and will be 0.0% on single processor systems. **swait** is the time a processor spends "spinning" while waiting for a spinlock. **block** is the time a processor spends "blocked" while waiting for a kernel-level semaphore.

2-13. TEXT PAGE: uptime, w

The **uptime** command shows how long a system has been up, and who is logged in and what they are doing.

The **w** command is linked to **uptime** and prints the same output as **uptime -w**, displaying a summary of the current activity on the system.

Tool Source:	Standard UNIX (BSD 4.x)
Documentation:	man page
Interval:	on demand
Data Source:	Kernel registers/counters and /etc/utmp
Type of Data:	Global
Metrics:	Load averages, number of logged on users
Logging:	Standard output device
Overhead:	Varies, depending on number of users logged in
Unique Feature:	Easiest way to see time since last reboot, load averages
Full Pathname:	/usr/bin/uptime
Pros and Cons:	+ quick look at load average, how long systems been up - limited statistics

Syntax

```
uptime [-hlsuw] [user]

w [-hlsuw] [user]

-h  Suppress the first line and the header line
-l  Print long listing
-s  Print short listing
-u  Print only the utilization lines; do not show user information
-w  Print what each user is doing; same as w command.
```

Example

```
# uptime
11:23am up 3 days, 22:22, 7 users, load average: 0.67, 0.37, 0.30

# uptime -w
11:23am up 3 days, 22:22, 7 users, load average: 0.57, 0.37, 0.30
User      tty          login@    idle    JCPU    PCPU    what
root      console      9:26am   94:20           /usr/sbin/getty console
root      pts/0        9:26am    5           /sbin/sh
root      pts/3        9:26am   1:57           /sbin/sh
root      pts/4       10:16am    2           2    vi tools_notes
root      pts/5        9:43am           script
```


2-14. TEXT PAGE: `vmstat`

The `vmstat` command reports virtual memory statistics about processes, virtual memory, and CPU activity.

Tool Source:	Standard UNIX (BSD 4.x)
Documentation	man page, include files
Interval:	≥ 1 second
Data Source:	Kernel registers/counters
Type of Data:	Global
Metrics:	CPU, Memory
Logging:	Standard output device
Overhead:	Varies, depending on presentation interval
Unique Feature:	Cumulative VM statistics since last reboot
Full Pathname:	<code>/usr/bin/vmstat</code>
Pros and Cons:	+ minimal overhead - poorly documented - cryptic headings - lines wrap on 80-column character display - statistics can bleed together

Syntax

```
vmstat [-dns] [interval [count]]
vmstat -f | -s | -z

-d Include disk I/O information
-n Print in a format more easily viewed on a 80-column display
-S Include swapping information
-f Print number of processes forked since boot, number of pages used
  by all forked processes, and the average pages/forked process
-s Print virtual memory summary information
-z Zero the summary registers.
```

Key Metrics

The `vmstat` metrics include:

Process metrics

r	In run queue
b	Blocked for resource (I/O, paging, and so on)
w	Runnable or short sleeper (< 20 sec.) but swapped

VM metrics

avm	Active virtual pages
free	Number of pages on the free list
re	Page reclaims
at	Address translation faults
pi	Pages paged in
po	Pages paged out
fr	Pages freed by vhand, per second
sr	Pages surveyed (dereferenced) by vhand, per second

Fault metrics

in	Device interrupts per second
sy	System calls per second
cs	CPU context switch rate (switches/second)

CPU metrics

us	User mode utilization
sy	System mode utilization
id	Idle time

Examples

```
# vmstat -n 5 2
VM
      memory
      avm    free    re    at    pi    po    fr    de    sr    in    sy    cs
      7589    728      0      0      0      0      0      0      0    140    490    30

CPU
  cpu
  us sy id    r    b    w
  2  1 97      0    74      0
  7670    692      0      0      0      0      0    235    4959    170
  47 11 42      0    75      0

# vmstat -nS 5 2
VM
      memory
      avm    free    si    so    pi    po    fr    de    sr    in    sy    cs
      7984    584      0      0      0      0      0      0      0    140    490    30

CPU
  cpu
  us sy id    r    b    w
  2  1 97      0    75      0
  7972    545      0      0      0      0      0    203    462     53
  1  1 98      0    76      0

# vmstat -f
3949 forks, 497929 pages, average= 126.09
```

Module 2

Performance Tools

```
# vmstat -s
0 swap ins
0 swap outs
0 pages swapped in
0 pages swapped out
1116471 total address trans. faults taken
346175 page ins
7976 page outs
200675 pages paged in
16824 pages paged out
213104 reclaims from free list
216129 total page reclaims
110 intransit blocking page faults
587961 zero fill pages created
303212 zero fill page faults
248573 executable fill pages created
67077 executable fill page faults
0 swap text pages found in free list
80233 inode text pages found in free list
166 revolutions of the clock hand
106769 pages scanned for page out
13236 pages freed by the clock daemon
75633551 cpu context switches
1612387244 device interrupts
1137948 traps
247228805 system calls
```

2-15. SLIDE: Performance Monitoring Tools (HP Specific)

Performance Monitoring Tools (HP Specific)



	Global Metrics	Process Details	Alarming Capability
<code>glance</code>	Yes	Yes	Yes
<code>gpm</code>	Yes	Yes	Yes
<code>xload</code>	Yes	No	No



Student Notes

This slide shows the HP-specific, *run-time* performance monitoring tools included with HP-UX. Currently, `glance` and `gpm` are available for HP-UX. Both `glance` and `gpm` are optional, and can be purchased separately. If you are running 11i (any version), both `glance` and `gpm` are included with the Enterprise and Mission Critical Operating Environments.

The `glance` and `gpm` tools provide real-time monitoring capabilities specific to the HP-UX operating system. Both tools provide access to performance data not available with standard UNIX tools, and both tools use the `midaemon` (i.e. KI interface) to collect performance data, yielding much more accurate performance results.

`xload` is an X-windows application, which graphically shows the recent length of the CPU's run queue. It consists of a window that displays vertical lines which represent the average number of processes in the run queue over the previous intervals. The default interval size is 8 seconds.

2-16. TEXT: glance

The **glance** tool is available for HP-UX. This is the recommended (and preferred) performance monitoring tool for HP-UX systems (character-based display). This tool shows information that cannot be seen with any of the standard UNIX monitoring tools. The accuracy of the data is considered more reliable, as the source is the **midaemon**, as opposed to the kernel counters and registers.

NOTE: Free evaluation copies of **glance** and **gpm** can be obtained for trial periods. The phone number to obtain an evaluation copy is (800) 237-3990.

Tool Source:	HP
Documentation:	man page and on-line help
Interval:	>= 2 second
Data Source:	midaemon
Type of Data:	Global, Process, and Application
Metrics:	CPU, Memory, Disk, Network, and Kernel resources
Logging:	Standard output device, screen shots to a file
Overhead:	Varies, depending on presentation interval and number of processes
Unique Feature:	Per process (and global) system call rates Extensive on-line help for the metrics Sort by CPU usage, memory usage, or disk I/O usage Files opened per process
Full Pathname:	/opt/perf/bin/glance
Pros and Cons:	+ extensive per-process information + extensive global information + more accurate than standard UNIX tools - uses the "curses" display library - relatively slow startup - not bundled with the OS (prior to 11i)

Syntax

```
glance [-j interval] [-p {dest}] [-f dest] [-maxpages numpages]
        [-command] [-nice nicevalue] [-nosort] [-lock]
        [-adviser_off] [-adviser_only] [-bootup]
        [-iterations count] [-syntax filename] [-all_trans]
        [-all_instances] [-disks <n>] [-kernel <path>] [-nfs <n>]
        [-pids <n>] [-no_fkeys]
```

Key Metrics

The glance tool includes reports for the following areas:

Hot Key	GLANCE PLUS REPORT	FUNCTION
a	CPU by Processor	All CPUs Performance Stats
c	CPU Report	CPU Utilization Stats
d	Disk Report	Disk I/O Stats
g	Process List	Global Process Stats
h		Help
i	I/O by Filesystem	I/O by Filesystem
l	Network by LAN	Lan Stats
m	Memory Report	Memory Stats
n	NFS Report	NFS Stats
s	Process selection	Single process information
t	System Table Report	OS Table Utilization
u	Disk report	Disk Queue Length
v	I/O by Logical Volume	Logical Volume Mgr Stats
w	Swap Detail	Swap Stats
z		Zero all Stats
A	Application List	
B	Global Waits	
D	DCE Activity	
F	Process Open Files	
G	Process Threads	
H	Alarm History	
I	Thread Resource	
J	Thread Wait	
K	DCE Process List	
L	Process System Calls	
M	Process Memory Regions	
N	NFS Global Activity	
P	PRM Group List	
R	Process Resources	
T	Transaction Tracker	
W	Process Wait States	
Y	Global System Calls	
Z	Global Threads	
?		Help with options
<CR>		Update screen with new data

See Module 3 for a more complete discussion of glance and gpm.

2-17. TEXT PAGE: gpm

The **gpm** tool is a graphical version of **glance**. All the benefits of using **glance** apply to **gpm** (GlancePlus Monitor).

NOTE: Free evaluation copies of **Glance** and **gpm** can be obtained for a 90-day trial period. The phone number to obtain the evaluation copy is (800) 237-3990.

Tool Source:	HP
Documentation:	man page and on-line help
Interval:	>= 1 second
Data Source:	midaemon
Type of Data:	Global, Process, Application
Metrics:	CPU, Memory, Disk, Network, kernel resources
Logging:	Standard output device and screen shots to a file
Overhead:	Varies, depending on presentation interval and number of processes
Unique Feature:	Alarming capabilities Performance advisor
Full Pathname:	/opt/perf/bin/gpm
Pros and Cons:	+ extensive per-process information + extensive global information + more accurate than standard UNIX tools - no selection for printing graphs - not bundled with the OS, prior to Ili

Syntax

```
gpm [-nosave] [-rpt <rptname>] [-sharedcbr] [-nice <nicevalue>] [-lock]
    [-disks <n>] [-kernel <path>] [-lfs <n>] [-nfs <n>] [-pids <n>]
    [xoptions]
```

Glance and GPM Advantages

Both Glance and GPM:

- Use the same metrics
- Use the **midaemon** and kernel registers/counters as data sources
- Have adjustable presentation intervals
- Have the ability to **renice** processes
- Provide alarming capability (via /var/opt/perf/advisor.syntax)
- Provide per-CPU metrics

- Can be configured to monitor application performance (that is, groups of processes)

Glance Advantages

Advantages of using **Glance** include:

- It is independent of X-Windows.
- It uses less overhead.

GPM Advantages

Advantages of using **gpm** include:

- It has customizable advisor syntax, which generates color-coded alarms.
- Has the ability to **kill** processes
- Reports are customizable.
- More comprehensive online documentation is available.

See Module 3 for a more complete discussion of glance and gpm.

2-18. TEXT PAGE: xload

xload is a graphical tool that will display the average length of the run queue over recent 10 second intervals. Since it is displayed in its own window on a graphics terminal, the window can be resized to accommodate good detail and many intervals at once.

Tool Source:	HP
Documentation:	man page
Interval:	10 seconds (default)
Data Source:	Kernel registers
Type of Data:	Global
Metrics:	Run queue length
Logging:	none
Overhead:	Very little
Unique Feature:	Visual representation of run queue lengths
Full Pathname:	<code>/usr/contrib./bin/X11/xload</code>
Pros and Cons:	+ visual representation of run queue lengths in real time + expandable window for greater time and detail + self-scaling - no scale labels - no per-processor information

Syntax

```
xload [ -toolkitoption ... ] [ -scale integer ] [ -update seconds ]  
      [ -hl | highlight color ] [ -jumpscroll pixels ] [ label string ]  
      [ -nolabel ] [ -lights ]
```

Example

```
xload -update 30
```

2-19. SLIDE: Data Collection Performance Tools (Standard UNIX)

Data Collection Performance Tools (Standard UNIX)				
	Global Metrics	Process Details	Alarming Capability	
<code>acctcom</code>	Some	Some	No	
<code>sar</code>	Yes	No	No	

Student Notes

This slide shows the standard UNIX *data collection* tools included with HP-UX. Data collection tools gather performance data and other system-activity information, and store this data to a file on the system.

By default, not too many standard UNIX tools perform data collection. The two most common tools are the `acct` (system accounting) suite of tools and `sar` (via the `sadc` and `sa1` programs), the system activity reporter.

2-20. TEXT PAGE: acct Programs

The system accounting programs are primarily a financial tool and are designed to charge for time and resources used on the system. Information such as connect time, pages printed, disk space used for file storage, and commands executed (and the resources used by those commands) is collected and stored by the acct commands. Generally not considered a performance tool, the accounting commands can provide useful data for certain situations.

Description

Tool Source:	Standard UNIX (System V)
Documentation:	man pages
Interval:	on demand
Data Source:	Kernel registers and other kernel routines
Type of Data:	System resources used, on a per user basis
Metrics:	Connect time, Disk space used, others
Logging:	Binary file <code>/var/adm/acct/pacct</code>
Overhead:	Medium to large (up to 33%), depending on number of users and amount of activity
Unique Feature:	Shows the amount of system resources being consumed by each user on the system. Logs every command executed by every user on the system.
Full Pathname:	<code>/usr/sbin/acct/[acct_command]</code>
Pros and Cons:	+ provides information to charge users for system use + extensive system utilization information kept - extremely large overhead, especially on an active system. - poor documentation

Syntax

```
/usr/sbin/acct/acctdisk  
  
/usr/sbin/acct/acctdusg [-u file] [-p file]  
  
/usr/sbin/acct/accton [file]  
  
/usr/sbin/acct/acctwtmp reason  
  
/usr/sbin/acct/closewtmp  
  
/usr/sbin/acct/utmp2wtmp  
  
and many more ...
```

System Accounting Notes

- System Accounting can be started:

Manually

Run the `/usr/sbin/acct/startup` command.

Automatically at Boot Time

Edit the `/etc/rc.config.d/acct` file and set the `START_ACCT` parameter equal to one (for example, `START_ACCT=1`).

- Only terminated processes are reported.
- Accounting reports include:
 - CPU time accounting
 - Disk accounting
 - Memory accounting
 - Connect time accounting
 - User command history
 - Several more

2-21. TEXT PAGE: sar

The **sar** tool comes with additional programs, which assist in performance data collection and storage. The performance data is kept for one month before being overwritten with new data. Since collected data is overwritten each month, monitoring the files' sizes is unnecessary.

The **sadc** program is a data collector which runs in the background, usually started by **sar** or **sa1**.

The **sa1** program is a convenient shell script for collecting and storing "**sar**" data to a log file under **/var/adm/sa**. This script is typically run from root's cron file and collects (by default) three system snapshots per hour.

The **sa2** program is also a convenient shell script for converting collected sar data (binary format) into readable ASCII report files. The report files are typically stored in **/var/adm/sa**. The **sa2** script is also normally run from root's cron file.

Tool Source:	Standard UNIX (System V)
Documentation:	man page
Interval:	>= 1 second
Data Source:	Kernel registers
Type of Data:	Global
Metrics:	CPU, Disk, Kernel resources
Logging:	Binary file under /var/adm/sa
Overhead:	Varies, depending on snapshot interval
Unique Feature:	Only standard UNIX <i>performance</i> data collector
Full Pathname:	/usr/sbin/sar
Pros and Cons:	+ familiarity + relatively low overhead - no per process information - accuracy not as good as MeasureWare/●VPA

Syntax

```
sar [-u|d|ycwaqvmAMS] [-o file] t [n]
sar [-u|d|ycwaqvmAMS] [-s time] [-e time] [-i sec] [-f file]
```

Some data collection related options:

-s	The start time of the desired data
-e	The end time of the desired data
-i	The size of the reporting interval in seconds
-o	The file to write the data to
-f	The file to read the data from

Configure Data Collection through cron Jobs

To set up **sar** data collection, add the following to **root's** cron file:

```
0 * * * 0,6 /usr/sbin/sa/sa1
0 8-17 * * 1-5 /usr/sbin/sa/sa1 1200 3
0 18-7 * * 1-5 /usr/sbin/sa/sa1
5 18 * * 1-5 /usr/sbin/sa/sa2 -s 8:00 -e 18:01 -i 3600 -u
5 18 * * 1-5 /usr/sbin/sa/sa2 -s 8:00 -e 18:01 -i 3600 -b
5 18 * * 1-5 /usr/sbin/sa/sa2 -s 8:00 -e 18:01 -i 3600 -g
```

Create the **/var/adm/sa** directory:

```
mkdir /var/adm/sa
```

Some systems recommend adding the above entries to **adm's** cron file instead of **root's**. On these systems, be sure to give write access to all users on the **/var/adm/sa** directory.

```
chmod a+w /var/adm/sa
```

sur /var/opt/perf/parm : remplir le nom du serveur
id = Nom du serveur

on achete Reserach Windows qui va collecter des
fichiers de /var/opt/perf/datafiles et qui va les
lors

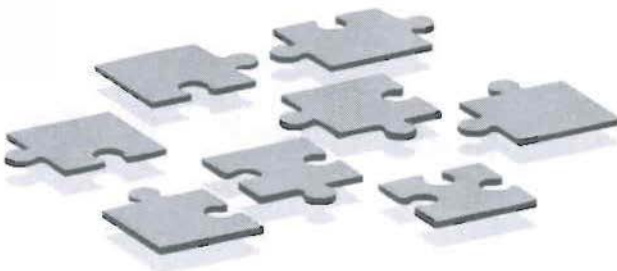
~~mv~~ mwa status
stop
start

2-22. SLIDE: Data Collection Performance Tools (HP-Specific)

Data Collection Performance Tools (HP-Specific)



	Global Metrics	Process Details	Alarming Capability
MeasureWare/OVPA	Yes	Yes	Yes
PerfView/OVPM	Yes	Yes	Yes
Data Source Integration	User Definable	User Definable	User Definable



Sas Windows

Student Notes

This slide shows the HP-specific *data collection* performance tools, which can be added to an HP-UX system. The MeasureWare/OVPA (OpenView Performance Agent) and PerfView/OVPM (OpenView Performance Manager) tools are available for HP-UX systems. These tools are optional products (separately purchasable).

These tools significantly enhance a customer's ability to track performance trends and review historical performance data about a system. The standard UNIX tools collect little to no per-process information, and have no alarming capabilities. With the MeasureWare/OVPA and PerfView/OVPM tools, global *and* per-process information is collected. In addition, *alarms* can be set to notify a user when a collected metric exceeds a defined threshold.

Recently, PerfView was renamed OpenView Performance Manager and MeasureWare was renamed OpenView Performance Agent. There were no other significant changes made to the products.

2-23. TEXT PAGE: MeasureWare/OVPA and DSI Software

MeasureWare/OVPA is the recommended and preferred tool for collecting performance data on an HP-UX system. MeasureWare/OVPA collects all the global and process statistics, consolidates the data into a 5-minute summary, and writes the record to a circular log file. Processes can be grouped into applications, and various thresholds are available for determining which processes are included in the summary.

Ovpa version 3.x is identical to Measureware. Ovpa version 4.x serves the same purpose, but has a new user interface.

Included with MeasureWare/OVPA is a product/tool called Data Source Integration (DSI). DSI allows custom, application-specific metrics to be defined and collected via the MeasureWare/OVPA product. This custom information can include database statistics, networking statistics collected with NetMetrix, or MIB information from a networking device (router or gateway) collected with SNMP.

Tool Source:	HP
Documentation:	man pages, manual, on-line help
Interval:	1 minute and 5 minute summaries
Data Source:	midaemon
Type of Data:	Global, Process, Application
Metrics:	CPU, Memory, Disk, Network, Other
Logging:	Circular binary files under <code>/var/opt/perf/datafiles</code>
Overhead:	Number of processes and number of application definitions
Unique Feature:	Parameter file to define the extent of data collection. Circular, compact log file format
Full Pathname:	<code>/opt/perf/bin/mwa</code>
Pros and Cons:	+ extensive global information + extensive per-process information + customize data collection with DSI - requires another tool (PerfView/OVPM) for graphical analysis - not included with the base OS

Syntax

```
mwa [action] [subsystem] [parms]
```

in which *action* is

start	Start all or part of MeasureWare/OpenView Performance Agent. (default)
stop	Stop all or part of MeasureWare/OpenView Performance Agent.
restart	Reinitialize all or part of MeasureWare/OpenView Performance Agent. This option causes some processes to be stopped and restarted.

Module 2

Performance Tools

status List the status of all or part of MeasureWare/OpenView Performance Agent processes.

MeasureWare/OVPA and Data Source Integration Notes

- The MeasureWare/OVPA agent for HP-UX is part of the RPM (Resource and Performance Management) set of performance tools. To find the complete list of available RPM products, visit the RPM Web site at: www.hp.com/go/rpm
- MeasureWare/OVPA is designed for use with the PerfView/OVPM Analyzer tool and features extensive alarming syntax.
- The utility and extract programs for MeasureWare/OVPA provide many features for the analysis and management of the MeasureWare/OVPA log files.
- The MeasureWare/OVPA agent is fully integrated with the OpenView product line and is capable of sending alarm messages to the PerfView/OVPM Monitor, Network Node Manager, and IT Operations.
- The MeasureWare/OVPA agent is available for a large number of UNIX platforms including: AIX, Solaris, NCR System VR4, Microsoft Windows NT, and more.
- Data Source Integration (DSI) is one of the most powerful features of MeasureWare/OVPA. DSI provides the ability to log data from any data source – as long as it writes its output to **stdout**.
- HP sells additional agents, which make use of this data source integration to allow for the monitoring of databases, network operating systems (for example, Windows NT and NetWare), and the Network Response Monitoring metrics (a facility of NetMetrix).
- Data can be imported from such operating environments as SAP/R3 and Baan.

See the course B5136S – “Performance Management with HP OpenView” for a more complete discussion of MeasureWare/OVPA.

2-24. TEXT PAGE: PerfView/OVPM

The PerfView/OVPM tool allows collected MeasureWare/OVPA information to be viewed in a feature-rich, GUI interface. Graphs, charts, alarms, and other details are easily viewed with the PerfView/OVPM tool. Similarly to the MeasureWare product, OVPM version 3 is identical to PerfView, whereas OVPM version 4 has the same functionality but has a new user interface.

Tool Source:	HP
Documentation:	Man pages, manual, online help
Interval:	On demand
Data Source:	MeasureWare/OVPA log files
Type of Data:	Global, Process, and Application
Metrics:	CPU, Memory, Disk, Network, others
Logging:	To central monitoring workstation
Overhead:	number of systems being analyzed, number of systems sending alarms
Unique Feature:	Many predefined graph templates. Access to any system currently running the MeasureWare/OVPA agent.
Full Pathname:	/opt/perf/bin/pv
Pros and Cons:	+ Centralized and automated performance monitoring + Can view data from DSI sources + Graphs can be saved in a worksheet format - Does not come standard with the OS

Syntax

```
pv [options]
```

PerfView/OVPM Notes

There are three components that make up the PerfView/OVPM product:

PerfView/OVPM Analyzer

- The PerfView/OVPM Analyzer allows for the performance administrator to easily access data from any MeasureWare/OVPA Agent.
- By default, the last 8 days of data are pulled in to be analyzed, but any amount of data that has been collected can be retrieved.
- The PerfView/OVPM Analyzer allows you to compare multiple systems against a specific metric as well for load balancing.
- The graphs produced by the PerfView/OVPM Analyzer can be stored, or printed out to any Postscript or PCL printer.
- As with all of the RPM products, the PerfView/OVPM Analyzer is fully integrated with Network Node Manager and IT Operations.

Module 2
Performance Tools

PerfView/OVPM Monitor

- The PerfView/OVPM Monitor receives alarms sent by MeasureWare/OVPA agents.
- It allows you to filter alarms by severity and type.
- The PerfView/OVPM Monitor is an optional module and may *not* be required if you are also running Network Node Manager or IT Operations.

PerfView/OVPM Planner

- The PerfView/OVPM Planner allows you to use collected MeasureWare/OVPA data to see performance trends.
- The more data provided to the PerfView/OVPM Planner and the less time you project it, the more accurate the reports will be.
- The PerfView/OVPM Planner is not a true capacity-planning tool in that it does not provide modeling or simulation capability.

See the course B5136S – “Performance Management with HP OpenView” for a more complete discussion of PerfView/OVPM.

2-25. SLIDE: Network Performance Tools (Standard UNIX)

Network Performance Tools (Standard UNIX)



	Resource	Super User Access Required
<code>netstat</code>	Various LAN Statistics	No
<code>nfsstat</code>	Network File Sharing Statistics	No
<code>ping</code>	Test Network Connectivity and Packet Round-Trip Response Time	No

Student Notes

This slide shows the standard UNIX *networking* performance tools included with HP-UX. Networking performance tools monitor performance and errors on the network.

The standard UNIX networking tools primarily allow for monitoring of performance. The HP-specific tools will introduce the ability to tune some networking parameters to better meet the needs of a system's networking environment.

NOTE: Super user (or root) access is *not* needed to monitor networking status by default.

2-26. TEXT PAGE: netstat

The **netstat** command displays general networking statistics. Information displayed includes:

- active sockets per protocol
- network data structures (like route tables)
- LAN card configuration and traffic

Tool Source:	Standard UNIX (BSD 4.x)
Documentation:	man pages and manual
Interval:	on demand
Data Source:	Kernel registers and LAN card
Type of Data:	Global
Metrics:	Network, LAN I/O, Sockets
Logging:	Standard output device
Overhead:	Varies, depending on network activity
Unique Features:	Shows established and listening sockets. Shows traffic going through LAN interface card. Shows amount of memory allocated to networking
Full Pathname:	/usr/bin/netstat
Pros and Cons:	+ provides lots of information on networking configuration - provides lots of metrics; not all metrics are documented well

Syntax

```
netstat [-aAn] [-f address-family] [system [core]]
netstat [-f address-family] [-p protocol] [system [core]]
netstat [-gin] [-I interface] [interval] [system [core]]
```

Examples

Display network connections

```
# netstat -n
Active Internet connections
Proto Recv-Q Send-Q Local Address           Foreign Address         (state)
tcp        0      0 156.153.192.171.1128    156.153.192.171.1129    ESTABLISHED
tcp        0      0 156.153.192.171.1129    156.153.192.171.1128    ESTABLISHED
tcp        0      0 156.153.192.171.947     156.153.192.171.1105    ESTABLISHED
Active UNIX domain sockets
Address Type  Recv-Q Send-Q Inode    Conn    Refs    Nextref Addr
c61300 dgram  0      0 814afc    0        0        0 /var/tmp/psb_front_socket
c87e00 dgram  0      0 914c4c    0        0        0 /var/tmp/psb_back_socket
de4f00 stream 0      0 0         0 f75280    0        0
f71200 stream 0      0 0         0 f75280    0        0 /var/spool/sockets/X11/0
```

Display network interface information:

```
# netstat -in
Name  Mtu  Network      Address           Ipkts    Ierrs    Opkts    Oerrs    Coll
ni0*  0    none         none              0         0         0         0         0
ni1*  0    none         none              0         0         0         0         0
```



```

100  4608  127          127.0.0.1          6745  0  6745  0  0
lan0 1500 156.153.192.0 156.153.192.171 156  0  u  0  0

```

Display network interface traffic:

```

# netstat -I lan0 5
(lan0)-> input      output      (Total)-> input      output
      packets      packets      packets      packets
      188          172          6973         6785
      2            1            2            1
      .
      .
      .

```

Display protocol status:

```

# netstat -s
tcp:
    2244 packets sent
        1191 data packets (217208 bytes)
        4 data packets (5840 bytes) retransmitted
        692 ack-only packets (276 delayed)
        318 control packets
    2277 packets received
        1188 acks (for 195140 bytes)
        144 duplicate acks
        1160 packets (236775 bytes) received in-sequence
        0 completely duplicate packets (0 bytes)
        83 out-of-order packets (0 bytes)
        0 discarded for bad header offset fields
        0 discarded because packet too short
    134 connection requests
    120 connection accepts
    243 connections established (including accepts)

udp:
    0 bad checksums
    164 socket overflows
    0 data discards

ip:
    460730 total packets received
    0 bad header checksums
    0 with ip version unsupported
    2253 fragments received
    2670 packets not forwardable
    0 redirects sent

icmp:
    1989 calls to generate an ICMP error message
    0 output histogram:
        echo reply: 727
        destination unreachable: 1989
    727 responses sent

arp:
    0 bad packet lengths
    0 bad headers

probe:
    0 packets with missing sequence number
    0 memory allocations failed

igmp:
    0 messages received with bad checksum
    10939700 membership queries received
    10969833 membership queries received with incorrect field(s)
    0 membership reports received

```

2-27. TEXT PAGE: `nfsstat`

The `nfsstat` command displays network file system (NFS) statistics. Categories of NFS information include:

- server statistics
- client statistics
- RPC statistics
- performance detail statistics

Tool Source:	Sun Microsystems
Documentation:	nman pages
Interval:	on demand
Data Source:	Kernel registers
Type of Data:	Global
Metrics:	NFS, RPC
Logging:	Standard output device
Overhead:	Varies, depending on NFS activity
Unique Feature:	Shows RPC calls, retransmissions, and timeouts.
Full Pathname:	<code>/usr/bin/nfsstat</code>
Pros and Cons:	+ reports both client and server activity - limited documentation

Syntax

```
nfsstat [ -cmnrsz ]
```

Examples

To reset all `nfsstat` counters to zero:

```
# nfsstat -z
```

To display server/client RPC and NFS statistics:

```
# nfsstat (this defaults to nfsstat -cnrs)
```

```
Server rpc:
Connection oriented:
calls      badcalls  nullrecv  badlen    xdr call  dupchecks  dupreqs
0          0          0         0         0         0          0

Connectionless oriented:
calls      badcalls  nullrecv  badlen    xdr call  dupchecks  dupreqs
0          0          0         0         0         0          0
```

Server nfs:

calls badcalls
0 0

Version 2: (0 calls)

null	getattr	setattr	root	lookup	readlink	read
0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%
wrocache	write	create	remove	rename	link	symlink
0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%
mkdir	rmdir	readdir	statfs			
0.0%	0.0%	0.0%	0.0%			

Version 3: (0 calls)

null	getattr	setattr	lookup	access	readlink	read
0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%
write	create	mkdir	symlink	mknod	remove	rmdir
0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%
rename	link	readdir	readdir+	fsstat	fsinfo	pathconf
0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%
commit						
0.0%						

Client rpc:

Connection oriented:

calls	badcalls	badxids	timeouts	newcreds
20	0	0	0	0
badverfs	timers	cantconn	nomem	interrupts
0	17	0	0	0

Connectionless oriented:

calls	badcalls	retrans	badxids	timeouts	waits	newcreds
20	0	0	0	0	0	0
badverfs	timers	toolbig	nomem	cantsend	buflocks	
0	17	0	0	0	0	

Client nfs:

calls	badcalls	clgets	cltoomany
20	0	20	0

Version 2: (20 calls)

null	getattr	setattr	root	lookup	readlink	read
0.0%	18.90%	0.0%	0.0%	0.0%	0.0%	0.0%
wrocache	write	create	remove	rename	link	symlink
0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%
mkdir	rmdir	readdir	statfs			
0.0%	0.0%	1.5%	1.5%			

Version 3: (0 calls)

null	getattr	setattr	lookup	access	readlink	read
0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%
write	create	mkdir	symlink	mknod	remove	rmdir
0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%
rename	link	readdir	readdir+	fsstat	fsinfo	pathconf
0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%
commit						
0.0%						

2-28. TEXT PAGE: ping

The `ping` command sends an ICMP echo packet to a host, and times how long it takes for the echo packet to return. This command is often used to test connectivity to another system. Specific details of the implementation include:

- An ICMP echo packet is sent once a second.
- Upon receipt of the echo packet, the round-trip time is displayed.
- The ability to display (via the `-o` option) the IP route taken.

Tool Source:	Public Domain
Documentation:	man pages
Interval:	on demand
Data Source:	NIC and ICMP packets
Type of Data:	Network
Metrics:	Packet transmission
Logging:	Standard output device
Overhead:	minimal; one packet transmission per second
Unique Feature:	Shows round-trip times between systems Shows route taken to and from the second system.
Full Pathname:	<code>/usr/sbin/ping</code>
Pros and Cons:	+ familiarity + understood by all UNIX-based (and TCP/IP-based) systems - limited functionality

Syntax

```
ping [-oprvt] [-i address] [-t ttl] host [-n count]
```

Examples

Send two ICMP echo packets to host `star1`:

```
# ping star1 -n 2
PING star1: 64 byte packets
64 bytes from 156.153.193.1: icmp_seq=0. time=1. ms
64 bytes from 156.153.193.1: icmp_seq=1. time=0. ms

---star1 PING Statistics---
2 packets transmitted, 2 packets received, 0% packet loss
round-trip (ms)  min/avg/max = 0/0/1
```

Send one ICMP packet and display the IP path taken:

```
# ping -o 156.152.16.10 -n 1
PING 156.152.16.10: 64 byte packets
64 bytes from 156.152.16.10: icmp_seq=0. time=337. ms

----156.152.16.10 PING Statistics----
1 packets transmitted, 1 packets received, 0% packet loss
round-trip (ms)  min/avg/max = 337/337/337
1 packets sent via:
    15.63.200.2      - [ name lookup failed ]
    15.68.88.4       - [ name lookup failed ]
    156.152.16.1     - [ name lookup failed ]
    156.152.16.10    - [ name lookup failed ]

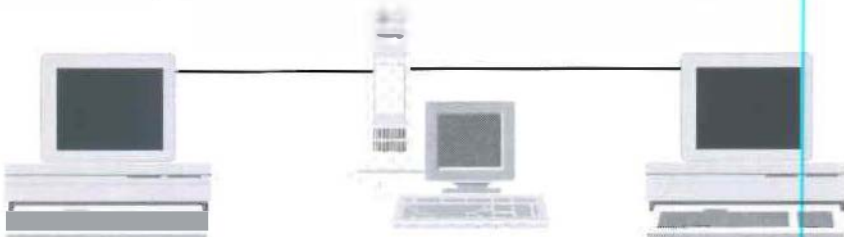
    15.68.88.43      - [ name lookup failed ]
    15.63.200.1      - [ name lookup failed ]
```

2-29. SLIDE: Network Performance Tools (HP-Specific)

Network Performance Tools (HP-Specific)



	Resource	Super User Access Required
lanadmin	Layer 2 Networking Statistics and NIC Reset	Yes
lanscan	LAN Hardware and Software Status	No
net tune (10.x)	Change Kernel Networking Parameters	Yes
ndd (11.x)	Change Kernel Networking Parameters	Yes
NetMetrix	Collects network performance data using RMON LAN probes	Yes



Student Notes

This slide shows the HP-specific *networking* performance tools included with HP-UX. The first three tools listed (**lanadmin**, **lanscan**, and **ndd/net tune**) come standard with the base OS. The NetMetrix product is an additional product.

The HP-specific networking tools display additional networking information and allow tuning of various networking parameters.

2-30. TEXT PAGE: lanadmin

The **lanadmin** command tests, displays statistics for, and allows modifications to LAN cards on the HP-UX system. Specific capabilities include:

- Resetting the LAN card and executing the LAN card self-tests
- Displaying and clearing LAN card statistics
- Changing the LAN card speed, the MTU size, and the link level address

Tool Source:	HP
Documentation:	man pages
Interval:	on demand
Data Source:	Kernel registers and Network Interface Card
Type of Data:	Network
Metrics:	Packet transmission status and errors
Logging:	Standard output device
Overhead:	minimal
Unique Feature:	Allows LAN interface card to be reset.
Full Pathname:	/usr/sbin/lanadmin
Pros and Cons:	+ provides extensive transmission statistics + allows for tuning of parameters normally requiring source code to change. - many statistics have little to no documentation

Syntax

```
/usr/sbin/lanadmin [-e] [-t]

/usr/sbin/lanadmin [-a] [-A station_addr] [-m] [-M mtu_size] [-R]
                  [-s] [-S speed] NetMgmtID

-e          Echo the input commands on the output device.

-t          Suppress the display of the command menu
            before each command prompt.
```

Example

```
# lanadmin

Test Selection mode.

lan      = LAN Interface Administration
menu     = Display this menu
quit     = Terminate the Administration
verbose  = Display command menu

Enter command: lan
```

Module2

Performance Tools

LAN Interface test mode. LAN Interface Net Mgmt ID = 4

```
clear      = Clear statistics registers
display    = Display LAN Interface status and statistics registers
end        = End LAN Interface Administration, return to Test Selection
menu      = Display the menu
ppa       = PPA Number of the LAN Interface
quit      = Terminate the Administration, return to shell
nmid      = Network Management ID of the LAN Interface
reset     = Reset LAN Interface to execute its selftest
specific  = Go to Driver specific menu
```

Enter command: display

```
Network Management ID      = 4
Description                = lan0 Hewlett-Packard LAN Interface Hw Rev 0
Type (value)              = ethernet-csmacd(6)
MTU Size                   = 1500
Speed                     = 10000000
Station Address            = 0x8000935c9bd
Administration Status (value) = up(1)
Operation Status (value)   = up(1)
Last Change               = 14465
Inbound Octets             = 3606105787
Inbound Unicast Packets    = 2767086
Inbound Non-Unicast Packets = 98379016
Inbound Discards           = 0
Inbound Errors             = 464396
Inbound Unknown Protocols  = 7114206
Outbound Octets            = 458391388
Outbound Unicast Packets   = 2842387
Outbound Non-Unicast Packets = 2874
Outbound Discards          = 0
Outbound Errors            = 0
Outbound Queue Length      = 0
Specific                   = 655367
```

Ethernet like Statistics Group

```
Index                    = 4
Alignment Errors         = 0
FCS Errors               = 0
Single Collision Frames  = 21353
Multiple Collision Frames = 42774
Deferred Transmissions   = 281589
Late Collisions          = 0
Excessive Collisions     = 0
Internal MAC Transmit Errors = 0
Carrier Sense Errors     = 0
Frames Too Long          = 0
Internal MAC Receive Errors = 0
```

2-31. TEXT PAGE: lanscan

The **lanscan** command displays the LAN card configuration and status. Items displayed include:

- Hardware address of LAN card slot
- Link level address of card
- Hardware status and interface status
- Other status and configuration information

Tool Source:	HP
Documentation:	man pages
Interval:	on demand
Data Source:	Network Interface Card
Type of Data:	Network
Metrics:	Interface status, Link Level Address
Logging:	Standard output device
Overhead:	minimal
Unique Feature:	Shows Link Level Address of system.
Full Pathname:	/usr/sbin/lanscan
Pros and Cons:	+ provides additional status information about network interface cards - no performance information

Syntax

```
lanscan [-ainv] [system [core]]

-a      Display station addresses only.  No headings.
-i      Display interface names only.  No headings.
-n      Display Network Management IDs only.  No headings.
-v      Verbose output.  Two lines per interface.  Includes displaying
        of extended station address and supported encapsulation methods.
```

Examples

Output from a 10.x system:

```
# lanscan
Hardware Station      Crd Hardware Net-Interface  NM  MAC      HP DLPI Mjr
Path      Address      In#  State      NameUnit  State  ID  Type      Support Num
2/0/2      0x080009D2C2DE 0    UP          lan0      UP     4   ETHER      Yes      52
```

Output from an 11.x system:

Module 2

Performance Tools

```
# lanscan
Hardware Station      Crd Hdw  Net-Interface  NM  MAC      HP-DLPI  DLPI
Path      Address      In# State NamePPA      ID  Type      Support  Mjr#
2/0/2      0x08000978BDB0 0    UP   lan0 snap0    1  ETHER     Yes      119
```

2-32. TEXT PAGE: `net tune` (HP-UX 10.x Only)

The `net tune` command allows modifications to be made to network parameters, which in previous releases were not modifiable. *This command was not included with any HP-UX 11.x release.* Parameters that can be modified with `net tune` include:

- arp configuration
- socket buffer sizes
- enable or disable IP forwarding

CAUTION: Use caution when making modifications with the tool. It is possible to hurt network performance severely or disable the LAN card when using this tool.

Tool Source:	HP
Documentation:	man pages, <code>net tune</code> help options (-?, -l, -h)
Interval:	on demand
Data Source:	Kernel registers and NIC
Type of Data:	Global
Metrics:	LAN tunable parameters
Logging:	Standard output device
Overhead	minimal
Unique Feature:	Change values of network parameters, which cannot otherwise be changed Change TCP send and receive buffer sizes without need for source code
Full Pathname:	/usr/contrib/bin/net tune
Pros and Cons:	+provides ability to modify networking behavior without needing source code + provides access to tunable parameters normally not available - can have a negative impact on performance if used the wrong way - minimal documentation

Syntax

```
net tune [-w] object [parm...]
net tune -h [-w] [object]
net tune -l [-w] [-b size] [object [parm...]]
net tune -s [-w] object [parm...] value...
```

-h (help) Print all information related to the object.
This information provides helpful hints about changing the value of an object.

-l (list) Print information regarding changing the value of object.

Module 2

Performance Tools

- s (set) Set object to value. An object may require more than one value.
- w Display warning messages (for example, 'value truncated'). These are normally discarded when the command is successful.

Examples

To get help information on all defined objects:

```
net tune -h
arp_killcomplete:
The number of seconds that an arp entry can be in the
completed state between references. When a completed arp
entry is unreferenced for this period of time, it is removed
from the arp cache.
. . .
```

To get help information on all TCP-related objects:

```
net tune -h tcp
tcp_receive:
The default socket buffer size in bytes for inbound data.
tcp_send:
The default socket buffer size in bytes for outbound data.
. . .
```

To set the value of the `ip_forwarding` object to 1:

```
net tune -s ip_forwarding 1
```

To get the value of the `tcp_send` object (socket send buffer size):

```
net tune tcp_send
```


2-33. TEXT PAGE: **ndd** (HP-UX 11.x Only)

The **ndd** command allows the examination and modification of several tunable parameters that affect networking operation and behavior. It accepts arguments on the command line or may be run interactively. The **-h** option displays all the supported and unsupported tunable parameter that **ndd** provides.

CAUTION: **ndd** was ported to HP-UX and contains references to some parameters that have not been implemented on the HP-UX O/S at this time. Reference the man page when in doubt. (Just because you can display a symbol's value and set it doesn't necessarily mean that the HP-UX kernel references the symbol!)

The **ndd** utility command accesses kernel parameters through the use of "pseudo device files". These pseudo device files are referred to as a *network device* on the **ndd** command line and selected from the following list:

/dev/arp	For ARP cache-related values
/dev/ip	For IP routing and forwarding parameters
/dev/rawip	Default IP time-to-live header value
/dev/tcp	Transport Connect Protocol (connection based) parameters
/dev/udp	User Datagram Protocol (connectionless) parameters

Tool Source:	HP
Documentation:	man pages, ndd -h (for help options)
Interval:	on demand
Data Source:	network device pseudo device files (reference above)
Type of Data:	Global
Metrics:	LAN tunable parameters
Logging:	Standard output device
Overhead	minimal
Unique Feature:	Change values of network parameters, which cannot otherwise be changed
Full Pathname:	/usr/bin/ndd
Pros and Cons:	+ provides ability to modify networking behavior without needing source code + provides access to tunable parameters normally not available - can have a negative impact on performance if used the wrong way - minimal documentation

Syntax

```
ndd -get network device parameter
```

Module 2

Performance Tools

```
ndd -set network device parameter
nnd -h    sup[ported]
nnd -h    unsup[ported]
nnd -h    [parameter]
nnd -c
```

At boot:

The file `/etc/rc.config.d/nndconf` contains tunable parameters that will be set automatically each time the system boots.

Examples

To list the contents of the "arp cache":

```
nnd -get /dev/arp arp_cache_report
```

To get help information on all supported tunable parameters:

```
nnd -h supported
```

To get a detail description of the tunable parameter, `ip_forwarding`:

```
nnd -h ip_forwarding
```

To get the current value of the tunable parameter, `ip_forwarding`:

```
nnd -get /dev/ip ip_forwarding
```

To set the value of the default TTL parameter for UDP to 128:

```
nnd -set /dev/udp udp_def_ttl 128
```

To re-read the configuration file, `/etc/rc.config.d/nndconf` without rebooting the system:

```
nnd -c
```

2-34. TEXT PAGE: **NetMetrix** (HP-UX 10.20 and 11.0 Only)

The **NetMetrix** product makes use of LAN probes to collect network traffic information. The LAN probes attach to the physical network and collect detailed information regarding the packets that pass through the probe.

Tools available with NetMetrix include:

- packet decoders
- network alarming capabilities
- reports including top packet generating systems
- data collection for trending

Tool Source:	HP
Documentation:	man pages, NRF (Network Response Facility) manual
Interval:	on demand
Data Source:	LAN probes
Type of Data:	LAN traffic
Metrics:	number of packets through cross-section of network
Logging:	NetMetrix binary file
Overhead:	Varies, depending on the number of LAN probes
Unique Feature:	Provides statistics regarding traffic on the entire network
Pros and Cons:	+ Statistics regarding total packet traffic - Additional cost - Requires LAN probes

NetMetrix Notes

- NetMetrix makes use of highly sophisticated devices (LAN probes) capable of collecting large amounts of detailed network information.
- NetMetrix is a truly distributed network management product that makes use of "mid-level managers" for data storage and alarming.
- There are a number of modules available with NetMetrix.
- NetMetrix's Internet Response Manager (IRM) and Internet Response Agent (IRA) fully integrate with HP OpenView products to provide a complete system and network management solution.

2-35. SLIDE: Performance Administrative Tools (Standard UNIX)

Performance Administrative Tools (Standard UNIX)		
	Resource	Super User Access Required
<code>ipcs</code>	List Semaphores, Message Queues, and Shared Memory Segments	No
<code>ipcrm</code>	Destroy Semaphores, Message Queues, and Shared Memory Segments	Yes
<code>nice</code>	Setting Process Priorities	Yes
<code>renice</code>	Modifying Process Priorities	Yes

Student Notes

This slide shows the standard UNIX *administrative* performance tools included with HP-UX. These tools are used to tune or modify system resources to better improve the performance of a system. These tools are typically used to *change* or *tune* a system's component, as opposed to *viewing* or *displaying* characteristics about the component.

Only the `root` user is allowed to use these commands, as making these modifications affects the performance for all users on the system.

NOTE: The `ipcs` program is really a performance-monitoring command; however, because it is usually run in conjunction with `ipcrm`, it is covered here to emphasize the relationship between the two commands.

2-36. TEXT PAGE: `ipcs`, `ipcrm`

The `ipcs` command displays information about active interprocess communication facilities. With no options, `ipcs` displays information in short format about message queues, shared memory segments, and semaphore sets that are currently active in the system.

The `ipcrm` command removes one or more specified message-queue, semaphore-set, or shared-memory identifiers.

Tool Source:	Standard UNIX (System V)
Documentation:	man pages
Interval:	on demand
Data Source:	Kernel registers
Type of Data:	Global, limited process
Metrics:	semaphore sets, message queues, shared memory
Logging:	Standard output device
Overhead:	varies, depending on the IPC resource in use
Unique Feature:	Shows the size, owner, and last user of message queues and shared memory segments.
Full Pathname:	<code>/usr/bin/ipcs</code> and <code>/usr/bin/ipcrm</code>
Pros and Cons:	+ shows orphan IPC entries + shows size of message queues and shared memory segments - process information limited to owner and last user

Syntax

```
ipcrm [-m shmid] [-q msqid] [-s semid]
```

```
ipcs [-mqs] [-abcopt] [-C corefile] [-N namelist]
```

- m Display information about active shared memory segments.
- q Display information about active message queues.
- s Display information about active semaphore sets.
- b Display largest-allowable-size information
- c Display creator's login name and group name
- o Display information on outstanding usage
- p Display process number information
- t Display time information

Module 2

Performance Tools

Examples

```
# ipcs -s
IPC status from /dev/kmem as of Fri Oct 17 12:56:36 1997
T      ID      KEY          MODE          OWNER      GROUP
Semaphores:
s       0 0x2f180002 --ra-ra-ra-   root      sys
s       3 0x412000a9 --ra-ra-ra-   root      root
s       4 0x00446f6e --ra-r--r--   root      root
s       6 0x01090522 --ra-r--r--   root      root
s       7 0x013d8483 --ra-r--r--   root      root
s      200 0x4c1c2f79 --ra-r--r--   daemon    daemon
```

```
# ipcrm -s 7
```

```
# ipcs -s
IPC status from /dev/kmem as of Fri Oct 17 12:57:42 1997
T      ID      KEY          MODE          OWNER      GROUP
Semaphores:
s       0 0x2f180002 --ra-ra-ra-   root      sys
s       3 0x412000a9 --ra-ra-ra-   root      root
s       4 0x00446f6e --ra-r--r--   root      root
s       6 0x01090522 --ra-r--r--   root      root
s      200 0x4c1c2f79 --ra-r--r--   daemon    daemon
```

ipcs -mob

2-37. TEXT PAGE: nice, renice

The **nice** command executes a command at a nondefault CPU scheduling priority. (The name is derived from being "nice" to other system users by running large programs at a weaker priority.)

The **renice** command alters the nice value of an existing process.

Tool Source:	Standard UNIX (System V)
Documentation:	man pages
Interval:	on demand
Data Source:	process table
Type of Data:	processes
Metrics:	priority
Logging:	standard output device
Overhead:	minimal
Unique Feature:	
Full Pathname:	/usr/bin/nice and /usr/bin/renice
Pros and Cons:	+ allows less important processes to run in the background + allows more important processes to run in the foreground - not an intuitive interface or syntax

Syntax

```
nice [-n newoffset_from_default_20] command [command_args]

renice [-n newoffset_from_current_value] [-g|-p|-u] id ...
```

An unsigned **newoffset** increases the system **nice** value for the command or process, causing it to run at a weaker priority. A negative value requires superuser privileges, and assigns a lower system **nice** value (stronger priority) to the process.

Examples

```
# ps -l
  F S      UID      PID  PPID  C PRI  NI       ADDR     SZ      WCHAN  TTY      TIME  CMD
  1 S          0    6044    6042  1 158  20     ff6680     85     87cec0 ttys2    0:00  sh
  1 R          0    8286    6044  6 179  30     1003d80    22             - ttys2    0:00  ps

# nice sh

# ps -l
  F S      UID      PID  PPID  C PRI  NI       ADDR     SZ      WCHAN  TTY      TIME  CMD
  1 S          0    6044    6042 11 158  20     ff6680     85     87cec0 ttys2    0:00  sh
  1 S          0    8290    8287  0 158  30     ff1680     85    100d3e0 ttys2    0:00  sh
  1 R          0    8293    8290  4 199  30     feae80     22             - ttys2    0:00  ps

# exit
```

Module 2

Performance Tools

```
# nice -10 sh
```

```
# ps -l
```

F S	UID	PID	PPID	C	PRI	NI	ADDR	SZ	WCHAN	TTY	TIME	COMD
1 S	0	6044	6042	0	158	20	ff6680	85	87cec0	ttyp2	0:00	sh
1 R	0	8297	8294	7	199	30	ff1280	22	-	ttyp2	0:00	ps
1 S	0	8294	6044	10	158	30	fea380	121	87e0c0	ttyp2	0:00	sh

```
# nice -5 ps -l
```

F S	UID	PID	PPID	C	PRI	NI	ADDR	SZ	WCHAN	TTY	TIME	COMD
1 S	0	6044	6042	0	158	20	ff6680	85	87cec0	ttyp2	0:00	sh
1 R	0	8304	8294	10	210	35	1003e80	22	-	ttyp2	0:00	ps
1 S	0	8294	6044	10	158	30	fea380	121	87e0c0	ttyp2	0:00	sh

```
# nice -n 30 sh
```

```
# ps -l
```

F S	UID	PID	PPID	C	PRI	NI	ADDR	SZ	WCHAN	TTY	TIME	COMD
1 S	0	6044	6042	0	158	20	ff6680	85	87cec0	ttyp2	0:00	sh
1 S	0	8305	8294	19	158	39	fb3300	121	87d6c0	ttyp2	0:00	sh
1 S	0	8294	6044	6	158	30	fea380	121	87e0c0	ttyp2	0:00	sh
1 R	0	8308	8305	4	220	39	feae80	22	-	ttyp2	0:00	ps

```
# exit
```

```
# nice -n 30 sh
```

```
# ps -l
```

F S	UID	PID	PPID	C	PRI	NI	ADDR	SZ	WCHAN	TTY	TIME	COMD
1 S	0	6044	6042	0	158	20	ff6680	85	87cec0	ttyp2	0:00	sh
1 S	0	8306	8294	1	158	30	f86200	121	87dc40	ttyp2	0:00	sh
1 S	0	8309	8306	7	158	0	fea380	121	87e0c0	ttyp2	0:00	sh
1 R	0	8312	8309	6	139	0	1003980	22	-	ttyp2	0:00	ps

2-38. SLIDE: Performance Administrative Tools (HP-Specific)

Performance Administrative Tools (HP-Specific)		
	Resource	Super User Access Required
getprivgrp	List system privileged groups	No
setprivgrp	Allocate special system privileges	Yes
rtprio	Set real time process priority (HP)	Privileged Access
rtsched	Set POSIX real time process priority	Privileged Access
scsictl	Set parameters on SCSI devices	Yes
serialize	Mark a program to run serially	Privileged Access
fsadm	Online JFS management tool	Yes
getext	Display JFS extent attributes	No
setext	Sets/changes JFS extent attributes	Yes
newfs	Create a file system	Yes
tunefs/vxtunefs	Change a file system's attributes	Yes
PRM/WLM	Process Resource Mgr/Work Load Mgr	Yes
WebQoS	Web Quality of Service	Yes

Student Notes

This slide shows the HP-specific *administrative* performance tools available on HP-UX systems. Many of the tools shown on the slide come standard with the base OS. The only tools that are add-on products are PRM, WLM, WebQoS, and Advanced JFS (**getext**, **setext**, and **fsadm**).

These HP-specific tools were developed to allow modifications and performance enhancements to the functionality unique to the HP-UX operating system.

2-39. TEXT PAGE: `getprivgrp`, `setprivgrp`

The `getprivgrp` command lists the access privileges of privileged groups.

The `setprivgrp` command sets the access privileges of privileged groups. If a `group_name` is supplied, access privileges are listed for that group only. The superuser is a member of all groups.

Access privileges include `RTPRIO`, `RTSCHED`, `MLOCK`, `CHOWN`, `LOCKRDONLY`, `SETRUGID`, `MPCTL`, `SPUCTL`, and `SERIALIZE`.

Tool Source:	HP
Documentation:	man pages
Interval:	on demand
Data Source:	/etc/group and kernel data structures
Type of Data:	users and groups
Metrics:	privilege access
Logging:	Standard output device
Overhead:	minimal
Unique Feature:	Gives non-root users access to privileges normally requiring root access.
Full Pathname	/usr/bin/getprivgrp and /usr/sbin/setprivgrp
Pros and Cons:	+ ability to assign additional privileges to groups - requires additional system management - cannot give privilege to a single user; must assign privileges to groups

Syntax

```
getprivgrp [-g|group_name]
```

```
setprivgrp [-g|groupname] [privileges]
```

-g Specify global privileges that apply to all groups.

Examples

```
# getprivgrp  
global privileges: CHOWN
```

```
# setprivgrp class CHOWN SERIALIZE RTPRIO
```

```
# getprivgrp  
global privileges: CHOWN  
class: RTPRIO CHOWN SERIALIZE
```

Notes

- Group privileges which can be modified are:

RTPRIO	Can use <code>rtprio()</code> call to set real-time priorities.
RTSCHED	Can use <code>sched_setparam()</code> call and <code>sched_setscheduler()</code> call to set POSIX.4 real-time priorities.
MLOCK	Can use <code>mlock()</code> to lock process text and data into memory, and the <code>shmctl()</code> <code>SHM_LOCK</code> function to lock shared memory segments
CHOWN	Can use <code>chown()</code> to change file ownership.
LOCKRDONLY	Can use <code>lockf()</code> to set locks on files that are open for reading only.
SETRUGID	Can use <code>setuid()</code> and <code>setgid()</code> to change, respectively, the real user ID and real group ID of a process.
SERIALIZE	Can use <code>serialize()</code> to force the target process to run serially with other processes that are also marked by this system call.
MPCTL	Can use <code>mpctl()</code> to lock a process or a thread to a specific processor on SMP systems. If processor sets are available, can be used to lock a process or a thread to a specific processor set.
SPUCTL	Can use <code>spuctl()</code> to enable and disable specific processors on SMP systems. (V-class, T-class, N-class, L-class, and Superdome only)

2-40. TEXT PAGE: rtprio

The **rtprio** command executes a specified command with a real-time priority, or changes the real-time priority of a currently executing process with a specific PID. Real-time priorities range from zero (strongest) to 127 (weakest).

Real-time processes are not subject to priority degradation and are considered of greater importance than all non-real-time processes.

CAUTION: Special care should be taken when using this command. It is possible to lock out other processes (including system processes) when using this command.

Tool Source:	HP
Documentation:	man pages
Interval:	on demand
Data Source:	process table
Type of Data:	process
Metrics:	process priority
Logging:	none
Overhead:	varies, depending on the activity of the process
Unique Feature:	assign real time priority to a process
Full Pathname:	<code>/usr/bin/rtprio</code>
Pros and Cons:	+ Can significantly improve the performance of a program - Can severely impact the performance of the system (if used incorrectly)

Syntax

```
rtprio priority command [arguments]  
rtprio priority -pid
```

```
rtprio -t command [arguments]  
rtprio -t -pid
```

```
-t execute command with a timeshare (non-real-time) priority, or  
change the currently executing process pid from a possibly  
real time priority to a timeshare priority.
```

Examples

Execute file `a.out` at a real-time priority of 100:

```
rtprio 100 a.out
```

Set the currently running process PID 24217 to a real-time priority of 40:

```
rtprio 40 24217
```


2-41. TEXT PAGE: `rtsched`

The `rtsched` executes commands with POSIX or HP-UX real-time priority, or changes the real-time priority of currently executing process `PID`.

All POSIX real-time priority processes are of greater scheduling importance than processes with HP-UX real-time or HP-UX timeshare priority. Neither POSIX nor HP-UX real-time processes are subject to degradation. POSIX real-time processes can be scheduled with one of three different POSIX scheduling policies specified: `SCHED_FIFO`, `SCHED_RR`, or `SCHED_RR2`. The number of POSIX real-time priority queues is tunable between the values of 32 and 512, and show up as a negative number between -1 and -512 when viewed with the `ps -ef` or `ps -el` commands.

CAUTION: Special care should be taken when using this command. It is possible to lock out other processes (including system processes) when using this command.

Tool Source:	HP
Documentation:	man pages (also see <code>rtsched(2)</code>)
Interval:	on demand
Data Source:	process table
Type of Data:	process
Metrics:	process priority
Logging:	none
Overhead:	varies, depending on the activity of the process
Unique Feature:	assign real time priority to a process
Full Pathname:	<code>/usr/bin/rtsched</code>
Pros and Cons:	+ Can significantly improve the performance of a program - Can severely impact the performance of the system (if used incorrectly)

Syntax

```

rtsched -s scheduler -p priority command [arguments]

rtsched | -s scheduler | -p priority -p pid

-s Specifies which scheduler to use, SCHED_FIFO (POSIX real-time),
  SCHED_RR (POSIX real-time), SCHED_RR2 (POSIX real-time),
  SCHED_RTRIO (HP-UX real-time), or SCHED_HPUX (HP-UX timeshare)

```

Module 2
Performance Tools

Examples

Execute file a.out at a POSIX real-time priority of 4:

```
rtsched -s SCHED_FIFO -p 4 a.out
```

Set the currently running process pid 24217 to a real-time priority of 20:

```
rtsched -s SCHED_RR -p 20
```

2-42. TEXT PAGE: `scsictl`

The `scsictl` command provides a mechanism for controlling a SCSI device. It can be used to query mode parameters, set configurable mode parameters, and perform SCSI commands. The operations are performed in the same order as they appear on the command line.

Tool Source:	HP
Documentation:	man pages
Interval:	on demand
Data Source:	SCSI disks
Type of Data:	disks
Metrics:	immediate reporting, I/O queue
Logging:	standard output device
Overhead:	minimal
Unique Feature:	Provides control over the behavior of an individual SCSI disk
Full Pathname:	<code>/usr/sbin/scsictl</code>
Pros and Cons	+ can improve performance by modifying the drive behavior - not all SCSI devices support the command - could misconfigure a disk, causing data to be lost in the event of a system crash

Syntax

```
scsictl [-akq] [-c command]... [-m mode[=value]]... device
```

`-a` Display the status of all mode parameters available.

`-m mode` Display the status of the specified mode parameter.

`immediate` For devices that support immediate reporting, this displays the immediate reporting status.

`queue_depth` For devices that support a queue depth greater than the system default, this mode controls how many I/Os the driver will attempt to queue to the device at any one time.

`-m mode=value` Set the mode parameter mode to value. The available mode parameters and values are listed above.

Module 2
Performance Tools

Examples

To display a list of all of the mode parameters, turn `immediate_report` on, and redisplay the value of `immediate_report`.

```
scsictl -a -m ir=1 -m ir /dev/rdisk/c0t6d0
```

will produce the following output:

```
immediate_report = 0; queue_depth = 8; immediate_report = 1
```

2-43. TEXT PAGE: **serialize**

The **serialize** command is used to force the target process to run serially with other processes also marked by this command. Once a process has been marked by **serialize**, the process stays marked until process completion, unless **serialize** is reissued.

Tool Source:	HP
Documentation:	man pages
Interval:	on demand
Data Source:	process table
Type of Data:	process
Metrics:	priority
Logging:	standard output device
Overhead:	minimal
Unique Feature:	decreases CPU and memory contention problems using standard functionality.
Full Pathname:	/usr/bin/serialize
Pros and Cons:	+ allows system to behave more efficiently when CPU and memory resources are scarce. - minimal documentation - only helps when CPU and memory resources are scarce

Syntax

```
serialize command [command_args]
```

```
serialize [-t] [-p pid]
```

-t Indicates the process specified by pid should be returned to timeshare scheduling.

Examples

Use **serialize** to force a database application to run serially with other processes marked for serialization. Type:

```
serialize database_app
```

Force a currently running process with a PID value of 215 to run serially with other processes marked for serialization. Type:

```
serialize -p 215
```

Return a process previously marked for serialization to normal timeshare scheduling. The PID of the target process for this example is 174. Type: **serialize -t -p 174**

2-44. TEXT PAGE: **fsadm**

The **fsadm** command is designed to perform selected administration tasks on HFS (10.20 or later) and JFS file systems. These tasks may differ between file system types.

For HFS file systems, **fsadm** allows conversions between **large** and **nolarge** files.

For VxFS file systems, **fsadm** allows file system resizing, extent (and directory) reorganization, and **large/nolarge** file conversions.

Tool Source:	Veritas and HP
Documentation:	man pages
Interval:	on demand
Data Source:	File System superblock and header structures
Type of Data:	file system header and data
Metrics:	fragmentation
Logging:	standard output device
Overhead:	Medium to large (up to 33%), depending on number of user and amount of activity
Unique Features:	Can defragment a file system, improving performance. (JFS) Can increase the size of a file system while it's mounted. (JFS)
Full Pathname:	/usr/sbin/fsadm
Pros and Cons:	+ provides greater manageability of file systems - many features (including defragmentation) are only available for JFS - requires purchasing the AdvanceJFS or OnlineJFS product.

Syntax

```
/usr/sbin/fsadm [-F vxfs|hfs] [-V] [-o largefiles|nolargefiles] mount_point|special  
/usr/sbin/fsadm [-F vxfs] [-V] [-b newsize] [-r rawdev] mount_point  
/usr/sbin/fsadm [-F vxfs] [-V] [-d] [-D] [-s] [-v] [-a days] [-t time]  
                [-p passes] [-r rawdev] mount_point
```

Examples

HFS Example

Convert a **nolargefiles** HFS file system to a **largefiles** HFS file system:

```
fsadm -F hfs -o largefiles /dev/vg02/lvol1
```

Display relevant HFS file system statistics:

```
fsadm -F hfs /dev/vg02/lvol1
```


JFS Example

Increase the size of the **var** file system to 100 MB while it is mounted and online:

```
lvextend -L 100 /dev/vg00/lvol7  
fsadm -F vxfs -b 102400 /var
```

Display fragmentation statistics for the **/home** file system:

```
fsadm -D -E /home
```

2-45. TEXT PAGE: **getext, setext**

The **getext** command displays extent attribute information of associated files on a JFS file system.

The **setext** command allows attributes related to JFS file systems and files within the JFS file system to be modified and tuned.

Tool Source:	Veritas
Documentation:	man pages
Interval:	on demand
Data Source:	JFS file system
Type of Data:	File system metadata structures
Metrics:	File system space allocation
Logging:	standard output device
Overhead:	minimal
Unique Feature:	Allows attributes of JFS files to be set
Full Pathname:	/usr/sbin/getext and /usr/sbin/setext
Pros and Cons:	+ can improve file system performance by modifying file attributes - require purchase of the Advanced JFS or Online JFS product

Syntax

```
/usr/sbin/getext [-V] [-f] [-s] file...
```

```
/usr/sbin/setext [-V] [-e extent_size] [-r reservation] [-f flag] file
```

Example

Display file attributes for the file, **file1**:

```
getext file1
```

```
file1:  Bsize 1024  Reserve 36  Extent Size 3  align noextend
```

The above output indicates a file with 36 blocks of reservation, a fixed extent size of 3 blocks, all extents aligned to 3-block boundaries, and the file cannot be extended once the current reservation is exhausted.

2-46. TEXT PAGE: newfs, tuneufs, vxtuneufs

The **newfs** command is a "friendly" front-end to the **mkfs** command. The **newfs** command calculates the appropriate parameters and then builds the file system by invoking the **mkfs** command.

The **tuneufs** command displays detailed configuration information for an HFS file system and allows some of the file system parameters to be modified.

Tool Source:	BSD 4.x, modified by HP, Veritas
Documentation:	man pages
Interval:	not applicable, on demand
Data Source:	file system header and superblock
Type of Data:	file system metadata structures
Metrics:	Block size, Fragment size, Minimum space
Logging:	standard output
Overhead:	minimal
Unique Feature:	Allows file system parameters to be displayed and set.
Full Pathname:	/usr/sbin/newfs, /usr/sbin/tuneufs, /usr/sbin/vxtuneufs
Pros and; Cons:	- File system parameters can be viewed and tuned for optimal performance - To tune many parameters, a re-initialization of the file system is required

Syntax

```
/usr/sbin/newfs [-F FStype] [-o specific_options] [-V] special

/usr/sbin/tuneufs [-A] [-v] [-a maxcontig] [-d rotdelay]
                  [-e maxbpg] [-m minfree] special-device

/usr/sbin/vxtuneufs
```

Notes

The initial file system parameters are set when the file system is first created with **newfs**. A small set of these parameters can be changed after the file system is created with **tuneufs**. **vxtuneufs** changes the attributes of the JFS file system when the file system is mounted.

NOTE: The **tuneufs** command works only for HFS file systems. The JFS file systems use other commands (**getext**, **setext**, **vxtuneufs**).

Module 2

Performance Tools

Examples

Create a file system on vg01 called rlvol1.

```
newfs -F hfs -b 16384 -f 2048 /dev/vg01/rlvol1
```

```
mkfs (hfs): Warning - 2 sector(s) in the last cylinder are not allocated.
mkfs (hfs): /dev/vg01/rlvol1 - 2048 sectors in 133 cylinders of 7 tracks, 22 sectors
21.0Mb in 9 cyl groups (16 c/g, 2.52Mb/g, 384 i/g)
Super block backups (for fsck -b) at:
    16, 2512, 5008, 7504, 10000, 12496, 14992, 17488, 19984
```

View the file system's configuration parameters:

```
tune2fs -v /dev/vg01/rlvol1
super block last mounted on:
magic 95014 clean FS_CLEAN time Fri Nov 28 07:02:58 1997
sblkno 8 cblkno 16 iblkno 24 dblkno 48
bsize 2048 cgroup 2048 cgroupsize 16 cgroupmask 0xffffffff8
ncg 9 size 10240 blocks 9858
bsize 16384 bshift 14 bmask 0xfffffc000
fsize 2048 fshift 11 fmask 0xfffff800
frag 8 fragshift 3 fsbtodb 1
minfree 10% maxbpg 38
maxcontig 1 rotdelay 0ms rps 60
csaddr 48 cssize 28672 csshift 10 csmask 0xfffffc00
ntrak 7 nsect 22 spc 154 ncyl 133
cpg 16 bpg 154 fpg 1232 ipg 384
nindir 4096 inopb 128 nsbf 2
nbfree 1230 ndir 2 nifree 3452 nffree 9
cgrator 0 fmod 0 ronly 0
fname fpack

cylinders in last group 5
blocks in last group 48
```

For VxFS file systems use:

```
# fsdb -F vxfs /dev/vg/NN/rlvolN
> 8192 B
> p S
```

2-47. **TEXT PAGE: Process Resource Manager (PRM)**

Process Resource Manager (PRM) allows the administrator to guarantee that important processes will receive the amount of memory, disk, and CPU time required to meet your performance objectives.

PRM works in conjunction with the standard HP-UX scheduler to improve response times for critical applications.

PRM provides state-of-the-art resource allocation that has long been missing in the UNIX environment.

Tool Source:	HP
Documentation:	PRM man pages (<code>prmconfig</code>)
Interval:	on demand
Data Source:	kernel registers and counters
Type of Data:	process groups as defined by the PRM configuration file.
Metrics:	CPU time, memory, and disk I/O bandwidth allocated to groups of processes
Logging:	standard output, <code>glance</code> , <code>gpm</code> , <code>perfvie</code> w/OVPM, <code>measureware</code> /OVPA
Overhead:	PRM only applies to time-shared processes. Real-time processes are not affected.
Unique Features:	allows the system administrator to control which groups of processes receive a certain percentage of the CPU's time, memory paging, and/or disk I/O request preference. CPU (per PRM group) entitlement and capping DISK (per PRM group per VG) entitlement Memory (per PRM group) entitlement, capping and selection method Application (per PRM group)
Full Pathname:	<code>/usr/sbin/prmconfig</code>
Pros and Cons:	+ Greater control of resource distributions - Optional product. Does not come standard with the OS. If you are running 11i in the Enterprise or Mission Critical Operating Environments, PRM is included.

See the course U5447S – “HP-UX Resource Management with PRM & WLM” for a more complete discussion of PRM.

2-48. TEXT PAGE: Work Load Manager (WLM)

The Work Load Manager sits on top of PRM and tunes it as necessary to meet the desired performance goals. The goals are defined in a configuration file in the form of Service Level Objectives (SLOs). The administrator defines these goals in the file and then lets WLM “tweak” PRM until either the goals are reached or they are attained as closely as possible.

Tool Source:	HP
Documentation:	WLM man pages (wlmd)
Interval:	on demand
Data Source:	kernel registers and counters
Type of Data:	process groups as defined by the WLM configuration file
Metrics:	As defined in the WLM configuration file
Logging:	Data can be sent to an EMS (Event Monitoring System)
Overhead:	Data collection of defined metrics and adjusting of PRM configuration
Unique Features:	allows the system administrator to define what Service Level Objectives are desired on the system and lets WLM to “tune” the system (via PRM) to obtain performance as close to those objectives as possible. CPU (per WLM group) entitlement DISK (per WLM group per VG) entitlement Memory (per WLM group) entitlement Application (per WLM group)
Full Pathname:	/opt/wlm/bin/wlmd
Pros and Cons:	+ Greater control of CPU distribution - Optional product. Does not come standard with the OS.

See the course U5447S – “HP-UX Resource Management with PRM & WLM” for a more complete discussion of WLM.

2-49. TEXT PAGE: Web Quality of Service — WebQoS

WebQoS manager is an example of the growing number of system performance management/enhancement products focused on specific server applications and environments.

The modern paradigm for application server management requires looking past simple performance metrics and forces us to start think a little out of the box. Do all requests received by a Web server warrant the same level of service? WebQoS allows the administrator to make decisions on the service level, based on several different criteria:

- admission control
- user differentiation
- activity differentiation
- application differentiation.

A discussion of the specifics of this product is beyond the scope of this class.

Tool Source: HP

Typical metrics: Number of concurrent users and response times.

Purpose: Maximize successful customer interactions and peak throughput.

Pros and Cons:

- Greater control of Web server resources tuned to specific client requests.
- Optional product. Does not come standard with the OS.

2-50. SLIDE: System Configuration and Utilization Information (Standard UNIX)

System Configuration and Utilization Information (Standard UNIX)				
	Resource	Portability		
bdf	Local and remote mounted file system space	Some		
df	Mounted file system space	Yes		
mount	Local and remote file system mounts	Yes		

Student Notes

This slide shows the standard UNIX tools for *displaying* system configuration and utilization information on an HP-UX system. System configuration and utilization tools are those which display configurations of LVM disks, file systems, and kernel resources.

2-51. TEXT PAGE: bdf, df

The **bdf** command displays the amount of free disk space available. If no file system is specified, the free space on all of the normally mounted file systems is printed. Free inode information can be displayed by using the **-i** option.

The **df** command displays the number of free 512-byte blocks and free inodes available for file systems by examining the counts kept in the superblock or superblocks. Blocks can be displayed in 1KB sizes by using the **-k** option.

Tool Source:	df , Standard UNIX (System V) bdf , Standard UNIX (Berkeley 4.x)
Documentation:	man pages
Interval:	on demand
Data Source:	File system superblocks
Type of Data:	Disk space resources
Metrics:	Disk space utilization
Logging:	Standard output
Overhead:	Minimal
Unique Feature:	Shows how much disk space is being utilized.
Full Pathname:	/usr/bin/bdf , /usr/bin/df
Pros and Cons:	+ Easy to use - minimal tuning statistics

Syntax

```
/usr/bin/bdf [-b] [-i] [-l] [-t type | [filesystem|file] ... ]
```

```
/usr/bin/df [-hefgiklnv] [-t|-P] [-o specific_options] [-V]
[special|directory]...
```

Examples — bdf Command

```
# bdf /usr
Filesystem      kbytes    used    avail %used Mounted on
/dev/vg00/lvol7 307200  279059    -9635  103% /usr

# bdf -i /
Filesystem      kbytes    used    avail %used    iused  ifree %iuse Mounted on
/dev/vg00/lvol3 40960    25093   14867    63%    3284   3960   45% /

# bdf -ib /home
Filesystem      kbytes    used    avail %used    iused  ifree %iuse Mounted on
dev/vg00/lvol4 53248    3586    46546    7%     513   12407    4% /home
Swapping        53248      0    40546    0%
3

# ll /home/paging
total 0
```

Module 2

Performance Tools

Examples — df Command

```
# df
/home      (/dev/vg00/lvol4) :    93062 blocks    12463 i-nodes
/opt       (/dev/vg00/lvol5) :   177124 blocks    23598 i-nodes
/tmp       (/dev/vg00/lvol6) :    90010 blocks    11982 i-nodes
/usr       (/dev/vg00/lvol7) :    52732 blocks     7011 i-nodes
/var       (/dev/vg00/lvol8) :   100122 blocks    13320 i-nodes
/stand     (/dev/vg00/lvol11) :    23596 blocks     5358 i-nodes
```

2-52. TEXT PAGE: mount

The **mount** command is used to mount file systems on the system. Other users can use **mount** to list mounted file systems. If **mount** is invoked without any arguments, it lists all of the mounted file systems from the file system mount table, **/etc/mnttab**.

Tool Source:	standard UNIX (System V)
Documentation:	man pages
Interval:	on demand
Data Source:	kernel mount table and /etc/mnttab file
Type of Data:	file system
Metrics:	file system type and mount options
Logging:	the file /etc/mnttab and standard output
Overhead:	minimal
Unique Feature:	used to mount HFS, JFS, and NFS file systems.
Full Pathname:	/sbin/mount
Pros and Cons:	+ displays valuable data regarding how file systems are mounted - different options depending on the type of file system being mounted

Syntax

```
/usr/sbin/mount [-l] [-p] [-v]
```

Examples

```
# mount -p
/dev/root      /          vxfs    log          0 0
/dev/vg00/lvol1 /stand    hfs     defaults    0 0
/dev/vg00/lvol6 /usr      vxfs    delaylog    0 0
/dev/vg00/lvol5 /tmp      vxfs    delaylog    0 0
/dev/vg00/lvol4 /opt      vxfs    delaylog    0 0
/dev/dsk/c0t4d0 /disk     hfs     defaults    0 0
/dev/vg00/lvol7 /var      vxfs    delaylog    0 0

# mount -v
/dev/root on / type vxfs log on Thu Sep 11 12:15:08 1997
/dev/vg00/lvol1 on /stand type hfs defaults on Thu Sep 11 12:15:11 1997
/dev/vg00/lvol6 on /usr type vxfs delaylog on Thu Sep 11 12:17:06 1997
/dev/vg00/lvol5 on /tmp type vxfs delaylog on Thu Sep 11 12:17:07 1997
/dev/vg00/lvol4 on /opt type vxfs delaylog on Thu Sep 11 12:17:07 1997
/dev/dsk/c0t4d0 on /disk type hfs defaults on Thu Sep 11 12:17:08 1997
/dev/vg00/lvol7 on /var type vxfs delaylog on Thu Sep 11 12:17:23 1997
#
```

2-53. SLIDE: System Configuration and Utilization Information (HP-Specific)

System Configuration and Utilization Information (HP-Specific)



	Resource	Portability
<code>diskinfo</code>	Size and model of local disk drives	No
<code>dmesg</code>	I/O tree and memory details	Some
<code>ioscan</code>	I/O tree and addressing	No
<code>vgdisplay</code>	Local volume group contents/attributes	No
<code>pvddisplay</code>	Local physical volume contents/attributes	No
<code>lvdisplay</code>	Local logical volume contents/attributes	No
<code>swapinfo</code>	Swap space utilization	No
<code>sysdef</code>	Sizes and values of kernel tables and parms	Some
<code>kmtune</code>	Query, set, or reset system parameters	Some
<code>kcweb</code>	Query, set, or reset system configuration	Some



Student Notes

This slide shows the HP-specific commands for displaying system configuration and utilization information. All the commands on the slide come standard with the base OS; none are add-on products.

These commands display the configuration and utilization of HP-specific subsystems. Many of these commands have corresponding commands on other UNIX systems that perform similar functions.

2-54. TEXT PAGE: diskinfo *Compte pas pour la réponse*

The **diskinfo** command determines whether the character special file named by **character_devicefile** is associated with a SCSI, CS/80, or Subset/80 disk drive. If so, **diskinfo** summarizes the disk's characteristics.

Both the size of disk and bytes per sector represent formatted media.

Tool Source:	HP
Documentation:	man pages
Interval:	on demand
Data Source:	controller on disk
Type of Data:	disk specific
Metrics:	disk capacity, sector size
Logging:	standard output
Overhead:	minimal
Unique Feature:	shows model number and manufacturer of disk
Full Pathname:	/usr/sbin/diskinfo
Pros and Cons:	+ can determine size and manufacturer of disk without having to open system - minimal tuning information

Syntax

si il met trop de temps donc un problème est survenue
`/usr/sbin/diskinfo [-b|-v] character_devicefile`

The **diskinfo** command displays information about the following characteristics of disk drives:

- vendor name, manufacturer of the drive (SCSI only)
- product identification number or ASCII name
- type, CS/80 or SCSI classification for the device
- size of disk specified in bytes
- sector size, specified as bytes per sector

Example

```
# diskinfo /dev/rdisk/c0t6d0
SCSI describe of /dev/rdisk/c0t6d0:
    vendor: QUANTUM
    product id: PD425S
    type: direct access
    size: 416575 Kbytes
    bytes per sector: 512
```

2-55. TEXT PAGE: **dmesg**

The **dmesg** command looks in a system buffer for recently printed diagnostic messages and prints them on the standard output. The messages are those printed by the system when unusual events occur (such as when system tables overflow or the file systems get full).

Tool Source:	HP
Documentation:	man pages
Interval:	on demand
Data Source:	kernel diagnostic buffer
Type of Data:	system diagnostic messages
Metrics:	kernel startup information
Logging:	standard output device
Overhead:	minimal
Unique Feature:	displays kernel diagnostic messages
Full Pathname:	<code>/sbin/dmesg</code>
Pros and Cons:	+ Allows kernel diagnostic messages to be recalled - Diagnostic messages can be lost since kernel buffer is a fixed size

Syntax

```
/usr/sbin/dmesg [-]
```

If the `-` argument is specified, **dmesg** computes (incrementally) the new messages since the last time it was run and places these on the standard output. This is typically used with **cron** (see **cron(1)**) to produce the error log `/var/adm/messages` by running the command:

```
/usr/sbin/dmesg - >> /var/adm/messages
```

every 10 minutes.

Example

```
# dmesg

Oct 17 12:39
vuseg=1815000
inet_cots:ok inet_cots:ok 1 graph3
2 bus_adapter
2/0/1 e720
2/0/1.0 tgt
2/0/1.0.0 stape
2/0/1.2 tgt
2/0/1.2.0 sdisk
2/0/1.3 tgt
2/0/1.3.0 stape
2/0/1.4 tgt
2/0/1.4.0 sdisk
2/0/1.7 tgt
2/0/1.7.0 setl
2/0/2 lan2
2/0/3 hi1
```

```
2/0/4 asio0
2/0/5 asio0
2/0/6 CentIf
2/0/7 c:720
2/0/7.5 tgt
2/0/7.5.0 sdisk
2/0/7.6 tgt
2/0/7.6.0 sdisk
2/0/7.7 tgt
2/0/7.7.0 sctl
2/0/8 audio
4 eisa
4/0/4 lan2
8 processor
9 memory
```

```
System Console is on the ITE
Networking memory for fragment reassembly is restricted to 36265984 bytes
Logical volume 64, 0x3 configured as ROOT
Logical volume 64, 0x2 configured as SWAP
Logical volume 64, 0x2 configured as DUMP
Swap device table: (start & size given in 512-byte blocks)
  entry 0 - major is 64, minor is 0x2; start = 0, size = 819200
Dump device table: (start & size given in 1-Kbyte blocks)
  entry 0 - major is 31, minor is 0x26000; start = 68447, size = 393217
Starting the STREAMS daemons.
B2352B HP-UX B.10.20 #1: Sun Jun  9 08:03:38 PDT 1996
```

```
Memory Information:
physical page size = 4096 bytes, logical page size = 4096 bytes
Physical: 393216 Kbytes, lockable: 302512 Kbytes, available: 349504 Kbytes
```

Using 1932 buffers containing 15360 Kbytes of memory.

```
SCSI: Request Timeout -- lbolt: 7543017, dev: cd000001
  lbp->state: 0
  lbp->offset: ffffffff
  lbp->uPhysScript: 2a24000
From most recent interrupt:
  TSTAT: 06, STST0: 04, STST1: 00, DSTAT: 80, DSPS: 00000006
lsp: 1feb00
  bp->b_dev: cd000001
  sb->io_id: 57b13
  sb->cdb: 08 00 00 08 00 00
  lbolt_at_timeout: 7544517, lbolt_at_start: 7543017
  lsp->state: 30d
  lsp->uPhysScript: 196e000
  lsp->upScript: 196a000
  lsp->upActivePtr: 196a000
  lsp->uActiveAdjust: 0
  lsp->upSavedPtr: 196a000
  lsp->uSavedAdjust: 0
  lsp->upPeakPtr: 196a000
  lsp->uPeakAdjust: 0
  lbp->owner: 1feb00
scratch_lbp: 0
Pre-DSP script dump [1b20020]:
  78051800 00000000 78030000 00000000
  00000002 02a24700 20000000 00000000
Script dump [1b20040]:
  9f0b0000 00000005 98080000 00000005
  98070000 00000001 58000008 00000000
```

2-56. TEXT PAGE: ioscanner

The **ioscanner** command scans system hardware, usable I/O system devices, or kernel I/O system data structures as appropriate, and lists the results. For each hardware module on the system, **ioscanner** displays the hardware path to the hardware module, the class of the hardware module, and a brief description.

By default, **ioscanner** scans the system and lists all reportable hardware found. The types of hardware reported include processors, memory, interface cards and I/O devices. Scanning the hardware may cause drivers to be unbound and others bound in their place in order to match actual system hardware. Entities that cannot be scanned are not listed.

On very large systems, **ioscanner** will operate much faster with the **-k** option. This will force **ioscanner** to read kernel structures built at boot time, rather than sending fresh inquiries to each hardware module.

Tool Source:	HP
Documentation:	man pages
Interval:	on demand
Data Source:	SCSI devices
Type of Data:	status and Hardware address
Metrics:	hardware status
Logging:	standard output
Overhead:	minimal
Unique Feature:	polls SCSI bus to retrieve status of SCSI devices
Full Pathname:	/usr/sbin/ioscanner
Pros and Cons:	+ Displays hardware addresses and corresponding device filenames. - Minimal performance data

Syntax

```
/usr/sbin/ioscanner [-k] -ul [-d driver] [-C class] [-I instance] [-H hw_path]
[-f | -n] [-F | -N] [device]
```

Examples

# ioscanner -f	Class	I	H/W Path	Driver	S/W State	H/W Type	Description
bc		0		root	CLAIMED	BUS_NEXUS	
graphics		0	0	graphi	CLAIMED	INTERFACE	Graphics
ba		0	2	bus_adapter	CLAIMED	BUS_NEXUS	Core I/O Adapter
ext_bus		0	2/0/1	c720	CLAIMED	INTERFACE	Built-in SCSI
target		0	3/0/1.0	tgt	CLAIMED	DEVICE	
disk		0	2/0/1.0.0	sflop	CLAIMED	DEVICE	TEAC FC-1 HF 07
target		1	2/0/1.1	tgt	CLAIMED	DEVICE	
tape		0	2/0/1.1.0	stape	CLAIMED	DEVICE	HP HP35470A
target		2	2/0/1.2	tgt	CLAIMED	DEVICE	
disk		1	2/0/1.2.0	sdisk	CLAIMED	DEVICE	TOSHIBA CD-ROM XM-3301TA
target		5	2/0/1.5	tgt	CLAIMED	DEVICE	
disk		4	2/0/1.5.0	sdisk	CLAIMED	DEVICE	QUANTUM FIREBALL1050S
target		6	2/0/1.6	tgt	CLAIMED	DEVICE	

Module 2 Performance Tools

```

disk      5  2/0/1 6.0  sdisk  CLAIMED  DEVICE  QUANTUM PD425S
target    7  2/0/1 7    tgt    CLAIMED  DEVICE
ctl       0  2/0/1 7.0  sct1   CLAIMED  DEVICE  Initiator
lan       0  2/0/2      lan2   CLAIMED  INTERFACE Built-in LAN
hil       0  2/0/3      hil    CLAIMED  INTERFACE Built-in HIL
tty       0  2/0/4      asio0  CLAIMED  INTERFACE Built-in RS-232C
tty       1  2/0/5      asio0  CLAIMED  INTERFACE Built-in RS-232C
ext_bus   1  2/0/6      CentIf CLAIMED  INTERFACE Built-in Parallel Interface
audio     0  2/0/8      audio  CLAIMED  INTERFACE Built-in Audio
processor 0  8          processor CLAIMED PROCESSOR Processor
memory    0  9          memory  CLAIMED  MEMORY  Memory

```

iosecan -fC disk

Class	I	H/W Patn	Driver	S/W State	H/W Type	Description
disk	5	2/0/1.6.0	sdisk	CLAIMED	DEVICE	QUANTUM PD425S

iosecan -fnc disk

Class	I	H/W Patn	Driver	S/W State	H/W Type	Description
disk	5	2/0/1.6.0	sdisk	CLAIMED	DEVICE	QUANTUM PD425S
			/dev/dsk/c0t6d0		/dev/rdisk/c0t6d0	

2-57. TEXT PAGE: **vgdisplay**, **pvdisplay**, **lvdisplay**

The **vgdisplay** command displays information about volume groups. If a specific **vg_name** is specified, information for just that volume group is displayed.

The **pvdisplay** command displays information about specific physical volumes (or disks) within an LVM volume group.

The **lvdisplay** command displays information about specific logical volumes within an LVM volume group.

Tool Source:	HP
Documentation:	man pages
Interval:	on demand
Data Source:	LVM header structures and <code>/etc/lvmtab</code>
Type of Data:	LVM configuration
Metrics:	mirroring, striping, other I/O policies
Logging:	standard output device
Overhead:	minimal
Unique Feature:	shows LVM configuration information
Full Pathname:	<code>/usr/sbin/vgdisplay</code> , <code>/usr/sbin/pvdisplay</code> , <code>/usr/sbin/lvdisplay</code>
Pros and Cons:	+ Only commands for viewing LVM configurations - Minimal tuning capabilities

Syntax

```
/sbin/vgdisplay [-v] [vg_name ...]  
/sbin/lvdisplay [-k] [-v] lv_path ...  
/sbin/pvdisplay [-v] [-b BlockList] pv_path ...
```

Examples

```
# vgdisplay  
--- Volume groups ---  
VG Name                /dev/vg00  
VG Write Access         read/write  
VG Status               available  
Max LV                 255  
Cur LV                 9  
Max PV                 16  
Cur PV                 2  
Max PE per PV          1016  
VGDA                   4  
PE Size (Mbytes)       4  
Total PE                726  
Alloc PE               279
```



```
Free PE          447
Total PVG        0

# pvdisplay /dev/dsk/c0t5d0
--- Physical volumes ---
PV Name          /dev/dsk/c0t5d0
VG Name          /dev/vg00
PV Status        available
Allocatable      yes
VGDA             2
Cur LV          7
PE Size (Mbytes) 4
Total PE         249
Free PE          0
Allocated PE     249
Stale PE         0
IO Timeout       default

# lvdisplay /dev/vg00/lvol1
--- Logical volumes ---
LV Name          /dev/vg00/lvol1
VG Name          /dev/vg00
LV Permission    read/write
LV Status        available/syncd
Mirror copies    0
Consistency Recovery MWC
Schedule         parallel
LV Size (Mbytes) 48
Current LE       12
Allocated PE     12
Stripes          0
Stripe Size (Kbytes) 0
Bad block        off
Allocation       strict/contiguous
```

2-58. TEXT PAGE: swapinfo

The **swapinfo** command prints information about device and file-system paging space. This information includes *reserved* space as well as *used* swap space.

NOTE: The term *swap* refers to an obsolete implementation of virtual memory; HP-UX actually implements virtual memory by way of paging rather than swapping. This command and others retain names derived from "swap" for historical reasons.

Tool Source:	HP
Documentation:	man pages
Interval:	on demand
Data Source:	kernel swap tables
Type of Data:	swap space
Metrics:	swap used, swap reserved, swap space configurations
Logging:	standard output device
Overhead:	minimal
Unique Feature:	Command can total all configured swap space into a one-line summary. Displays pseudoswap information (if configured).
Full Pathname:	/usr/sbin/swapinfo
Pros and Cons	+ provides valuable swap space configuration information - minimal documentation on pseudoswap

Syntax

```
/usr/sbin/swapinfo [-mtadfnrMqw]
```

Examples

```
# swapinfo -t
```

	Kb	Kb	Kb	PCT	START/	Kb			
TYPE	AVAIL	USED	FREE	USED	LIMIT	RESERVE	PRT	NAME	
dev	159744	19868	139876	12%	0		1	/dev/vg00/lvol2	
reserve	-	51220	51220						
memory	42112	15300	26812	36%					
total	201856	86388	115468	43%	-	0	-		

2-59. TEXT PAGE: sysdef

The **sysdef** command analyzes the currently running system and reports on its tunable configuration parameters.

Tool Source:	HP
Documentation:	man pages
Interval:	on demand
Data Source:	/stand/vmunix and the currently running kernel
Type of Data:	Tunable kernel parameters
Metrics:	Current configuration of kernel parameters
Logging:	Standard output device
Overhead:	Minimal
Unique Feature:	Shows current value and possible range of values
Full Pathname:	/usr/sbin/sysdef
Pros and Cons:	+ Shows current setting of kernel parameters - reboot required to change most parameters

Syntax

```
/usr/sbin/sysdef [kernel [master]]
```

Example

```
# /usr/sbin/sysdef
```

NAME	VALUE	BOOT	MIN-MAX	UNITS	FLAGS
acctresume	4	-	100-100		-
acctsuspend	2	-	-100-100		-
allocate_fs_swapmap	0	-	-		-
bufpages	2841	-	0-	Pages	-
create_fastlinks	0	-	-		-
dbc_max_pct	50	-	-		-
dbc_min_pct	5	-	-		-
default_disk_ir	1	-	-		-
dskless_node	0	-	0-1		-
eisa_io_estimate	768	-	-		-
eqmemsize	15	-	-		-
file_pad	10	-	0-		-
fs_async	0	-	0-1		-
hpux_aes_override	0	-	-		-
maxdsiz	16384	-	256-655360	Pages	-
maxfiles	60	-	30-2048		-
maxfiles_lim	1024	-	30-2048		-
maxssiz	2048	-	256-655360	Pages	-
maxswapchunks	256	-	1-16384		-
maxtsiz	16384	-	256-655360	Pages	-
maxuprc	75	-	3-		-
maxvgs	10	-	-		-
msgmap	255504	-	3-		-
nbuf	4788	-	0-		-
ncallout	292	-	6-		-

Module 2

Performance Tools

nodnode	150	-	-	-
ndilbuffers	30	-	1-	-
netisr_priority	-1	-	-1-127	-
netmemmax	5378048	-	-	-
nfile	800	-	14-	-
nflocks	200	-	2-	-
ninode	476	-	14-	-
no_lvm_disks	0	-	-	-
nproc	276	-	10-	-
npty	60	-	1-	-
nstrpty	60	-	-	-
nswapdev	10	-	1-25	-
nswapfs	10	-	1-25	-
public_shlibs	1	-	-	-
remote_nfs_swap	0	-	-	-
rtsched_numpri	32	-	-	-
sema	0	-	0-1	-
semmap	4128768	-	4-	-
shmem	0	-	0-1	-
shmuni	200	-	3-1024	-
streampipes	0	-	0-	-
swapmem_on	1	-	-	-
swchunk	2048	-	2048-16384	kbytes -
timeslice	10	-	-1-2147483648	Ticks -
unlockable_mem	801	-	0-	Pages -

Name - The name of the parameter
 Value - The current value of the parameter
 Boot - The value of the parameter at boot time
 Min - The minimum allowed value of the parameter
 Max - The maximum allowed value of the parameter
 Units - The units by which the parameter is measured
 Flags - Further describe the parameter
 M Parameter may be modified without rebooting

A comparable command, introduced at HP-UX 11.00, is **kmtune (1m)**.

2-60. TEXT PAGE: kmtune, kcweb

The **kmtune** command is used to query, set, or reset system parameters. **kmtune** displays the value of all system parameters when used without any options or with the **-S** or **-I** option. **kmtune** reads the master files and the system description files of the kernel and kernel modules. On Ili v2, **kmtune** is front-ended and will eventually be replaced entirely by **kctune**. **kctune** is part of a new, larger utility called **kcweb**.

Tool Source:	HP
Documentation:	man pages
Interval:	on demand
Data Source:	/stand/vmunix and the currently running kernel
Type of Data:	Tunable kernel parameters
Metrics:	Current configuration of kernel parameters
Logging:	Standard output device
Overhead:	Minimal
Unique Feature:	Works with dynamic and static kernel modules
Full Pathname:	/usr/sbin/kmtune

Syntax

```
/usr/sbin/kmtune [-l] [[-q name] . . .] [-S system file]
/usr/sbin/kmtune [[-s {+|=}value] . . .] [[-r name] . . .] [-S system file]
```

Examples

```
# /usr/sbin/kmtune
```

Parameter	Value
=====	=====
NSTRBLKSCHED	2
NSTREVENT	50
NSTRPUSH	16
NSTRSCHED	0

•
•
•

```
# /usr/sbin/kmtune -l -q maxdsiz
```

```
Parameter:  maxdsiz
Value:      0x04000000
Default:    0x04000000
Minimum:    -
Module:     -
Version:    - (Ili only)
Dynamic:    - (Ili only)
```

2-61. SLIDE: Application Profiling and Monitoring Tools (Standard UNIX)

Application Profiling and Monitoring Tools
(Standard UNIX)



	Resource	Super User Access Required
prof	Application Profiler	No
gprof	Enhanced Application Profiler	No
arm	Define and measure response time of transactions for an application	No

Student Notes

This slide shows the standard UNIX *application profiling* performance tools included with HP-UX. Application profiling tools provide in-depth details regarding the execution of a program, including the number of times each subroutine is called and the amount of time spent in each subroutine.

2-62. TEXT PAGE: **prof, gprof**

The **prof** and **gprof** tools are used to ascertain the library routines being called during the execution of a program.

The **prof** utility profiles the execution of an application by displaying the names of the routines being called, the number of times the different routines were called, and how much time was spent in each routine.

The **gprof** utility is an enhanced version of **prof**. It shows all the information available with **prof**, plus it displays a **call graph tree**, which details the call hierarchy of the routines. The **call graph tree** allows the parent routines, which called the children routines to be viewed.

Tool Source:	standard UNIX (System V)
Documentation:	man pages
Interval:	on demand
Data Source:	kernel routines called by the application
Type of Data:	function call flow
Metrics:	time spent in each function, number of times function was called
Logging:	binary file <code>mon.out</code>
●overhead:	significant delays in the execution of the application
Unique Feature:	shows the flow of the function calls
Full Pathname:	<code>/usr/bin/prof</code>
Pros and Cons:	+ shows where an application is spending its time - requires access to source code - requires application to be recompiled

Syntax

```
prof [-tcan] [-ox] [-g] [-z] [-h] [-s] [-m mdata] [prog]
gprof [options] [a.out [gmon.out...]]
```

Examples

```
cc -p prog.c -o program
./program
prof program
```

```
cc -G prog.c -o program
./program
gprof program
```

2-63. TEXT PAGE: Application Response Measurement (ARM) Library Routines

Description

The ARM library routines allow you to define and measure response time of transactions in any application that uses a programming language that can call a 'C' function.

The ARM library is named "libarm" and is provided in two versions, an archive version and a shared library version. It is strongly recommended that you use the shared (sometimes referred to as dynamic) library version.

In-depth discussion of this product is beyond the scope of this class.

NOTE: **arm** is a cross-platform tool and functionally replaces the **ttd** discussed in the next section. **glance** and **gpm** work equally well with either **arm** or **ttd**.

Documentation: `man 3 arm`

Interval: configurable

Platforms supported: HP-UX, IBM AIX, Sun Solaris, NCR

Pros and Cons: + Integrates with PerfView/MWA and other distributed management/monitoring tools.
- requires source code modification

Syntax: The six function calls used by **arm** are:

<code>arm_init</code>	Return a unique ID based on application and user.
<code>arm_getid</code>	Return a unique ID based on a transaction name.
<code>arm_start</code>	Mark the beginning of a specific transaction.
<code>arm_update</code>	Provide information or show progress of a specific transaction.
<code>arm_stop</code>	Mark the end of a specific transaction.
<code>arm_end</code>	Mark the end of an application.

2-64. SLIDE: Application Profiling and Monitoring Tools (HP-Specific)

Application Profiling and Monitoring Tools (HP-Specific)



	Description	Super User Access Required
<code>ttd</code>	Tracks how much time is spent between specific lines of code in a program	No
<code>caliper</code>	<code>caliper</code> is a runtime performance analyzer for programs compiled with C, C++ and Fortran 90 compilers on Itanium systems.	No



Student Notes

This slide shows some HP-specific application profiling tools included with HP-UX. Currently, the Transaction Tracker (`ttd`), and `caliper` are available for monitoring application behavior and performance.

In 10.20, there was a tool called `puma` which came with all standard programming language compilers (like C, Pascal, and Fortran). The `puma` tool allowed profiling data to be collected without having to modify the application source code, or recompiling the application (in many cases). `puma` has been excluded from the more recent releases of HP-UX.

The Transaction Tracker allows a programmer to time how long a program is spending within a certain area of code. The Transaction Tracker requires the source code be modified to include the starting point and the stopping point. The Transaction Tracker is included as part of the MeasureWare OVPA product. Transaction Tracker is HP-UX specific. `azm` (discussed earlier) is the generic version of the Transaction Tracker.

`caliper` is thread-aware, MP-aware, and features an easy command-line interface.

2-65. TEXT PAGE: Transaction Tracker

Description

The Transaction Tracker is a set of function calls that allow a programmer to time the execution of a particular body of code (referred to a "transaction"). The function calls are inserted into the source code to mark where a particular transaction begins and ends.

Glance and gpm can then be used to monitor how many times the transaction is called, and how long it takes for the transaction to complete.

Tool Source:	HIP
Documentation:	MeasureWare Users manual
Interval:	Every time Transaction Tracker function call is invoked within program
Data Source:	The ttd process
Type of Data:	Application execution times
Metrics:	Times to one hundredth of a second
Logging:	Binary file <code>/var/opt/perf/datafiles/logtrans</code>
Overhead:	Medium to large, depending on number of transactions being timed.
Unique Feature	Shows amount of time spent in a particular body of code
Full Pathname:	Function calls defined in <code>/opt/perf/include/tt.h</code>
Pros and Cons:	+ Integrated with glance and gpm; makes it easy to monitor how long transactions take. - Cannot be used within shell programs; C programs only (or programs which can call C routines).

Syntax

The four function calls used by Transaction Tracker are:

<code>tt_getid</code>	Names the transaction and returns a unique identifier.
<code>tt_start</code>	Signals the start of a unique transaction.
<code>tt_end</code>	Signals the end of the transaction.
<code>tt_abort</code>	Ends the transaction without recording times for the transaction.

2-66. TEXT PAGE: caliper — HP Performance Analyzer

Description

HP Caliper is a general-purpose performance analysis tool for applications on Itanium®-based HP-UX systems. HP Caliper allows you to understand the performance of your application and to identify ways to improve its run-time performance.

HP Caliper works with any Itanium-based binary and does not require your applications to have any special preparation to enable performance measurement.

The two primary ways to use HP Caliper are:

- As a performance analysis tool.
- As a profile based optimization (PBO) tool invoked by HP compilers.

The latest version of HP Caliper is available on the HP Caliper home page. You can find it at the <http://www.hp.com/go/hpcaliper/> site.

Overview

HP Caliper helps you dynamically measure and improve the performance of your native Itanium-based applications in three ways:

- Commands to measure the overall performance of your program.
- Commands to drill down to identify performance parameters of specific functions in your program.
- A simple way to optimize the performance of your program based on its specific execution profile.

HP Caliper does not require special compilation of the program being analyzed and does not require any special link options or libraries. HP Caliper selectively measures the processes, threads, and load modules of your application. An application's load modules are the main executable and all shared libraries it uses.

HP Caliper uses a combination of dynamic instrumentation of code and the performance monitoring unit (PMU) in the Itanium processor. HP Caliper uses the least-intrusive method available to gather performance data.

Supported Target Programs

HP Caliper includes support for:

- ✓ Programs compiled for Itanium- and Itanium 2-based systems. HP Caliper does not measure programs compiled for PA-RISC processors.
- ✓ Code generated by native and cross HP aC++, C++ and Fortran compilers, including inlined functions and C++ exceptions.
- ✓ Programs compiled with optimization or debug information, or both. This includes support for both the +objdebug and +noobjdebug options.

Module 2

Performance Tools

- Both ILP32 (+DD32) and LP64 (+DD64) programs, both 32-bit and 64-bit ELF formats.
- Archive-, minshared- or shared-bound executables.
- Both single- and multi-threaded applications, including MxN threads.
- Applications that `fork()` or `vfork()` or `exec()` themselves or other executables.
- Shell scripts and the programs they spawn.

Features

HP Caliper is simple to run because it uses a single command for all measurements. You specify the type of measurement and the target program as command-line arguments.

For example, to measure the total number of CPU cycles used by a program named `myprog`, just type:

```
caliper total_cpu myprog
```

HP Caliper features include:

- Multiple performance measurements, each of which can be customized through configuration files.
- All reports are available in text format, comma-delimited (CSV) format, and most reports are also available in HTML format for easier browsing.
- Performance data can be correlated to your source program by line number.
- Easy inclusion and exclusion of specific load modules, such as `libc`, when measuring performance.
- Both per-thread and aggregated thread reports for most measurements.
- Performance data reported by function, sorted to show hot spots.
- Support for multi-process selection capabilities.
- The ability to save performance data in files that you can use to aggregate data across multiple runs to generate reports without having to re-run HP Caliper.
- The ability to attach and detach to running processes for certain measurements.
- The ability to restrict PMU measurements to specific regions of your programs.
- Limited support for dynamically generated code.

2-67. SLIDE: Summary

Summary



- Different categories of performance tools
- Standard UNIX tools versus HP-specific tools
- Separately purchasable tools
- Kernel register-based tools versus midaemon-based Tools

Student Notes

To summarize this module, there are many performance tools for many different purposes. The objective of this module was to highlight all the performance tools available with HP-UX, to categorize them by function, and to describe how each tool worked.

In general, you should become most familiar with these tools:

```
sar
vmstat
top
glance/gpm (if available)
```

These tools will tend to be your most commonly used tools. Other tools will tend to be useful in more specialized situations.

Remember, never try to rely on just one tool to do everything. No tool will tell you everything. And every tool will mislead you somewhere down the line. No tool is perfect. That's why you need to be familiar with multiple tools.

2-68. LAB: Performance Tools Lab

Lab





Before we continue with a more focused discussion of `glance` and `gpm`, let's spend some time exploring the generic UNIX and HP-UX-specific tools discussed so far.

As you answer the following questions, try to categorize each tool as to its type and scope.

Student Notes

The goal of this lab is to gain familiarity with performance tools. A secondary goal is to get familiar with the metrics reported by the tools, although they will be explored in depth during the next days.

Directions

Set up: Change directories to: # `cd /home/h4262/tools`

Execute the setup script: # `./RUN`

Use `glance` (or `gpm` if you have a bit-mapped display), `sar`, `top`, `vmstat`, and any other available tools to answer the following questions. List as many as possible, and include the appropriate `OPTION` or `SCREEN`, which will give the requested information.

Specific numbers are not the important goal of this lab. The goal is to gain familiarity with a variety of performance tools.

Always investigate what the basic UNIX tools can tell you before running `glance` or `gpm`. You may want to run through this lab with the solution from the back of this book for more guidance and discussion.

1. How many processes are running on the system? Which tools can you use to determine this?
2. Are there any real time priority processes running? If so, list the name and priority. What tools can you use to determine this?
3. Are there any nice'd processes on the system? If so, list the name and priority for each. What tools can you use to determine this?
4. Are there any zombie processes on the system? If so, how many are there? What tools can you use to determine this?
5. What is the length of the run queue? What are the load averages? What tools can you use to determine this?

Module 2
Performance Tools

6. How many system processes are running? What tools can you use to determine this?

NOTE A system process is defined as a process whose data space is the kernel's data space (such as `swapper`, `vhand`, `statdaemon`, `unhashdaemon`, and `supsched`). `ps` reports their size as zero.

There are three ways this can be determined. If you get stuck on this question, move on. Don't spend more than a few minutes trying to answer this question.

7. What percentage of time is the CPU spending in different states? What tools can you use to determine this?

8. What is the size of memory?
What is the size of free memory?
What tools can you use to determine this?

9. What is the size of the swap area(s)?
What is the percentage of swap utilization?
What tools can you use to determine this?

10. What is the size of the kernel's in-core inode table?
How much of the inode table is utilized?
What tools can you use to determine this?

11. Are there any CPU-bound processes running (processes using a "lot" of CPU)?
If so, what is the name of the process?
What steps did you take to determine this?

12. Are there any processes running which are using a "lot" of memory? (A "lot" is relative, i.e. a large RSS size compared to other processes.)
If so, what is the name of the process?
What steps did you take to determine this?
Is memory utilization changing?

13. Are there any processes running which are doing any disk I/O? If so, what is the name of the process?
What steps did you take to determine this?
What are the I/O rates of the disk bound processes?
What files are open by this (these) process(es)?

NOTE: No processes are really doing a lot of physical disk I/O. However, `lab_proc3` is doing a LOT of logical I/O.

Module 2

Performance Tools

14. What is the current rate of semaphore or message queue usage? What tools can you use to determine this?
15. Is there any paging or swapping occurring? What tools can you use to determine this?
16. What is the system call rate? What tools can you use to determine this?
17. What is the buffer cache hit ratio? What tools can you use to determine this?
18. What is the tty I/O rate? What tools can you use to determine this?
19. Are there any traps (interrupts) occurring? What tools can you use to determine this?

20. What information can you collect about network traffic? What tools can you use to determine this?
21. What information can be gathered on CPUs in an SMP environment? What tools can you use to determine this?
22. What information can be gathered on Logical Volumes? What tools can you use to determine this?
23. What information can be gathered on Disk I/O? What tools can you use to determine this?
24. Shut down the simulation by entering:

./KILLIT

Module 2
Performance Tools

Module 3 — GlancePlus

Objectives

Upon completion of this module, you will be able to do the following:

- Compare GlancePlus with other performance monitoring/management tools.
- Start up the GlancePlus terminal interface (`glance`) and graphical user interface (`gpm`).

Si Runqer = 3 → 25%

Si cpu % > 25% → 25%

> 50% → 25%

> 75% → 25%

et si le cpu > 75% pendant 5 minutes
alors jaune

3-1. SLIDE: This Is GlancePlus

This is GlancePlus



Features

- Motif-based interface that offers exceptional ease-of-learning and ease-of-use
- State-of-the-art, award-winning on-line Help system.
- Rules-based diagnostics that use customizable system performance rules to identify system performance problems and bottlenecks.
- Alarms that are triggered when customizable system performance thresholds are exceeded.
- Tailor information gathering and display to suit your needs.
- Integrated into OpenView environments.

Capabilities

- Get detailed views of CPU, disk, and memory resource activity
- View disk I/O rates and queue lengths by disk device to determine if your disk loads are well balanced
- Monitor virtual memory I/O and paging
- Measure NFS activity
- And much more ...

Student Notes

GlancePlus is a performance monitoring diagnostic tool. GlancePlus software visually gives you the useful, accurate information you need to pinpoint potential or existing problems involving your system's CPU, memory, disk, or network utilization.

To help you monitor and interpret your system's performance data, GlancePlus software includes a rules-based adviser. Whenever threshold levels for measurements such as CPU utilization or disk I/O rates are exceeded the adviser notifies you with on-screen alarms. The adviser also applies rules to key performance measurements and symptoms and then gives you information to help you uncover bottlenecks or other performance problems.

NOTE: GlancePlus is integrated into OpenView Windows at the menu bar level.

GlancePlus offers a viewpoint into many of the critical resources that need to be measured in the open system environment.

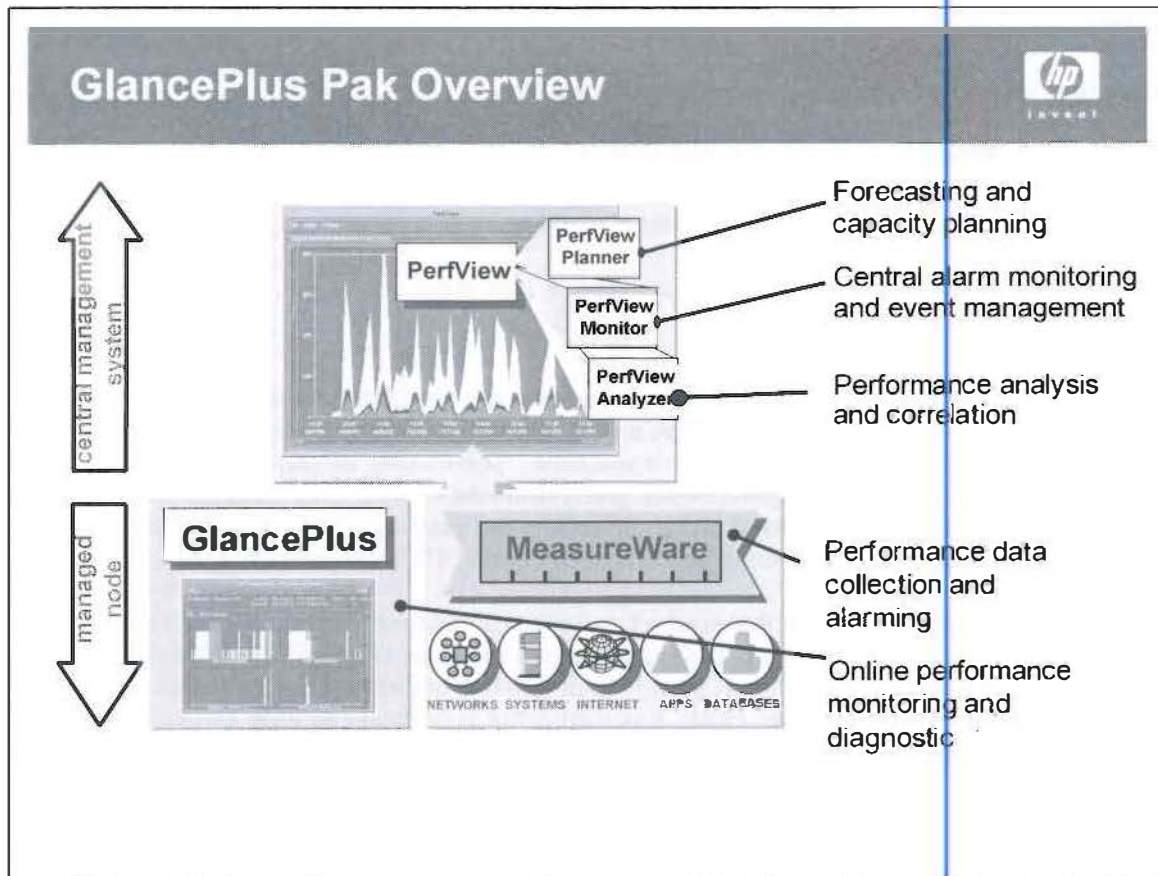
Benefits

- Save time and effort managing your system resources
- Better understand your computing environment
- Satisfy your end users' system performance needs quickly
- Leverage from a standard interface across vendor platforms

The features in the product yield a performance monitoring diagnostic solution that offers many benefits to the user. GlancePlus offers a tool that will make your analysis activities easier and quicker to perform. This will save you time. The display of various types of information will also allow you to get a better understanding of your own environment. The same GUI on the Motif version is used on all the supported platforms, which provides a leverage point for a standard user interface across several UNIX platforms.

Many times, just by cursory use of the product, people will discover certain things about their systems. You do not have to have a performance problem to use GlancePlus. This simple cursory use of the product has let many people gain a better understanding of their systems. This helps out when a problem does exist. Knowing what is normal can help identify what has become abnormal in your environment.

3-2. SLIDE: GlancePlus Pak Overview



Student Notes

The view here is from the heights.

For our purposes, we will focus our discussion on the capabilities of **glance** and **gpm** and the information and reports they can produce from a running HP-UX system. Also understand that GlancePlus may be used in conjunction with MeasureWare/OVPA to enhance and extend its capabilities. Many of you may have purchased **glance** in the GlancePlus Pak, which includes a license to run **glance**, **gpm** and to configure and run the MeasureWare/OVPA Agent (**mwa**) on your system.

The GlancePlus and MeasureWare/OVPA Agent products can be purchased separately or combined in the GlancePlus Pak. The Pak also includes (as of C.03.58.00 June 2002 application release) some event monitoring and graphical configuration components.

The components share a common measurement infrastructure, thus metrics, as well as applications have similar alarming mechanisms.

GlancePlus Pak

GlancePlus

Interfaces include:
`/opt/perf/bin/gpm`
`/opt/perf/bin/glance`

MeasureWare/OVPA

Interfaces include:
`/opt/perf/bin/extract`
`/opt/perf/bin/utility`

PerfView/OVPM

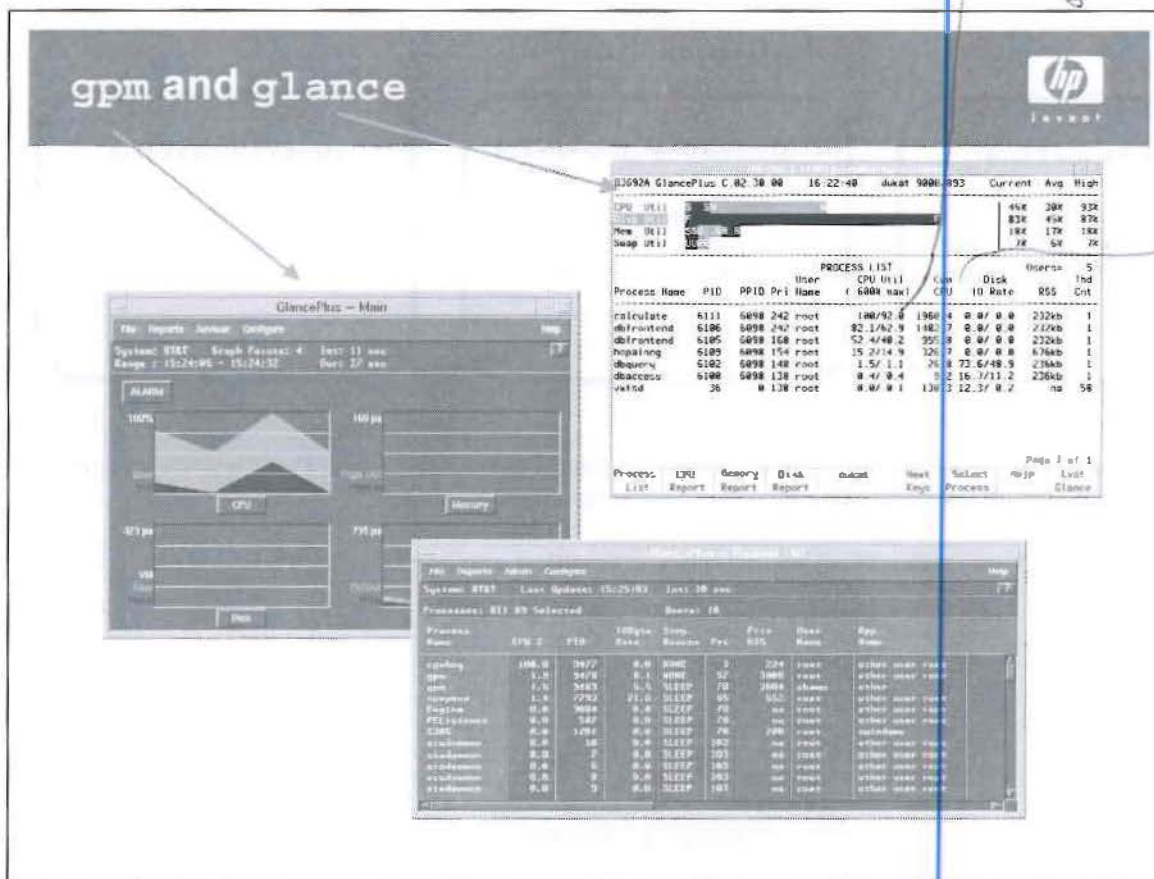
Interfaces include:
`/opt/perf/bin/pv`

Complete information on the configuration and use of MWA/OVPA and PerfView/OVPM are fully covered in the Hewlett-Packard Education Services' course:

PerfView MeasureWare (catalog number B5136).



3-3. SLIDE: gpm and glance



Student Notes

GlancePlus provides dual user interfaces:

The gpm GUI

See history of activity of the system with multiple window capability

Monitor your system while doing other work

Use alarms, symptoms and color to assist with monitoring

The glance Character Mode

Monitor performance remotely over slow datacom line

When no high resolution monitor is available

Creates less load on the system being monitored

Notes on starting the user interfaces: **gpm** and **glance**

Starting the GUI

gpm [options]

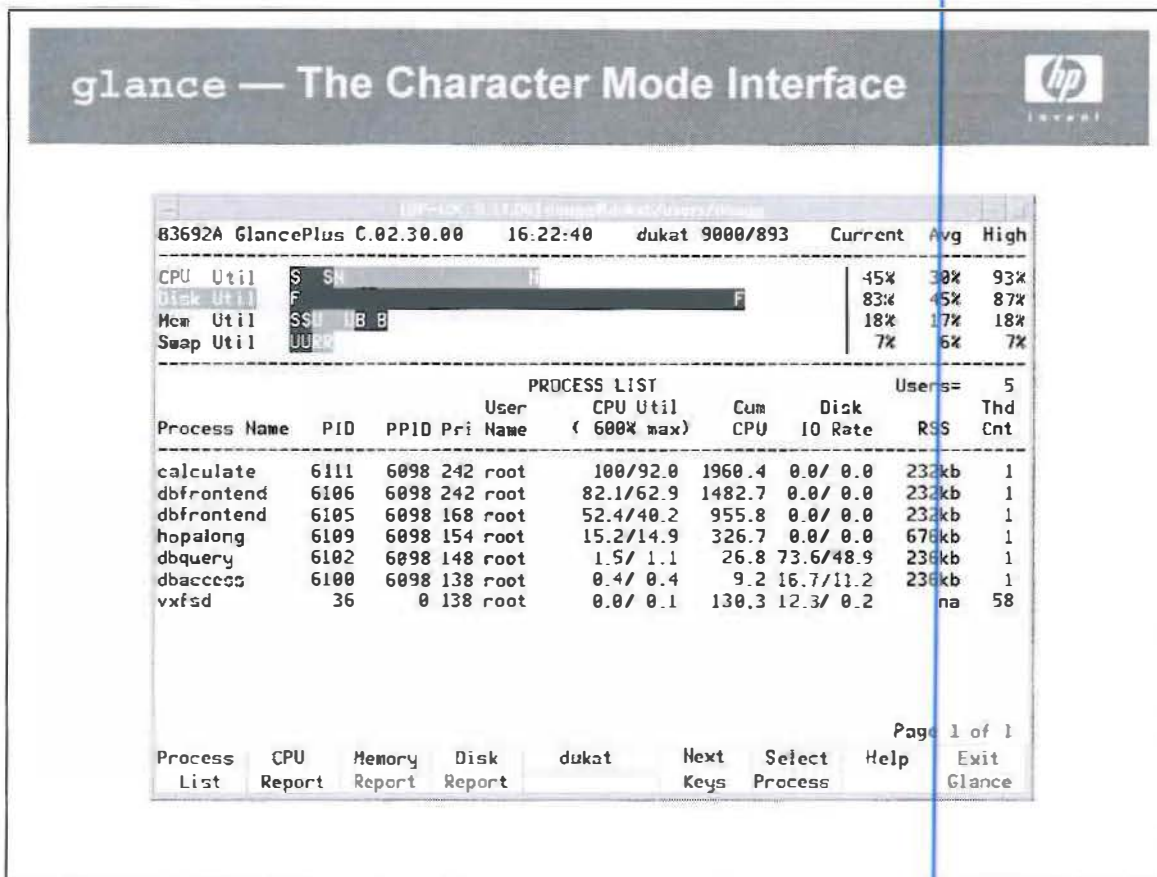
Starting the character based interface:

glance [options]

-nosave	Do not save the current configuration at the next exit
-rpt	Specify one or more additional report windows
-sharedclr	Share color scheme with other applications
-nice	Set the gpm nice value
Xoptions	Use X-Toolkit options such as -display

-j interval	Preset the number of seconds between screen refreshes
-p dest	Specify the continuous print option destination.
-lock	Allows glance to lock itself into memory
-nice	Set the glance nice value

3-4. SLIDE: glance — The Character Mode Interface



Student Notes

With **glance** you can run on almost any terminal or workstation, over a serial interface and relatively slow data communication links, and with lower resource requirements.

The default **Process List** screen is shown in the above screen capture, and provides general data on system resources and active processes. In addition, the user may "drill down" to more specific levels of detail in areas of CPU, memory, disk I/O, network, NFS system calls, swap, and system table screens.

Specific details on a per-process level are also available through the individual process screens.

For your convenience, the next two pages contain a hot key quick reference guide for the **glance** character mode interface.

Glance Hot Key Quick Reference

Top Level Screen Hot Keys

Hot Key	Screen Displayed/Description
a	CPU By Processor
c	CPU Report
d	Disk Report
g	Process List
i	I/O By File System
l	Network By Interface
m	Memory Report
n	NFS By System
t	System Tables Report
u	I/O By Disk
v	I/O By Logical Volume
w	Swap Space
A	Application List
B	Global Waits
D	DCE Global Activity
G	Process Threads
H	Alarm History
I	Thread Resources
J	Thread Wait
K	DCE Process List
N	NFS Global Activity
P	PRM Group List
T	Transaction Tracker
Y	Global System Calls
Z	Global Threads
?	Conunands Menu

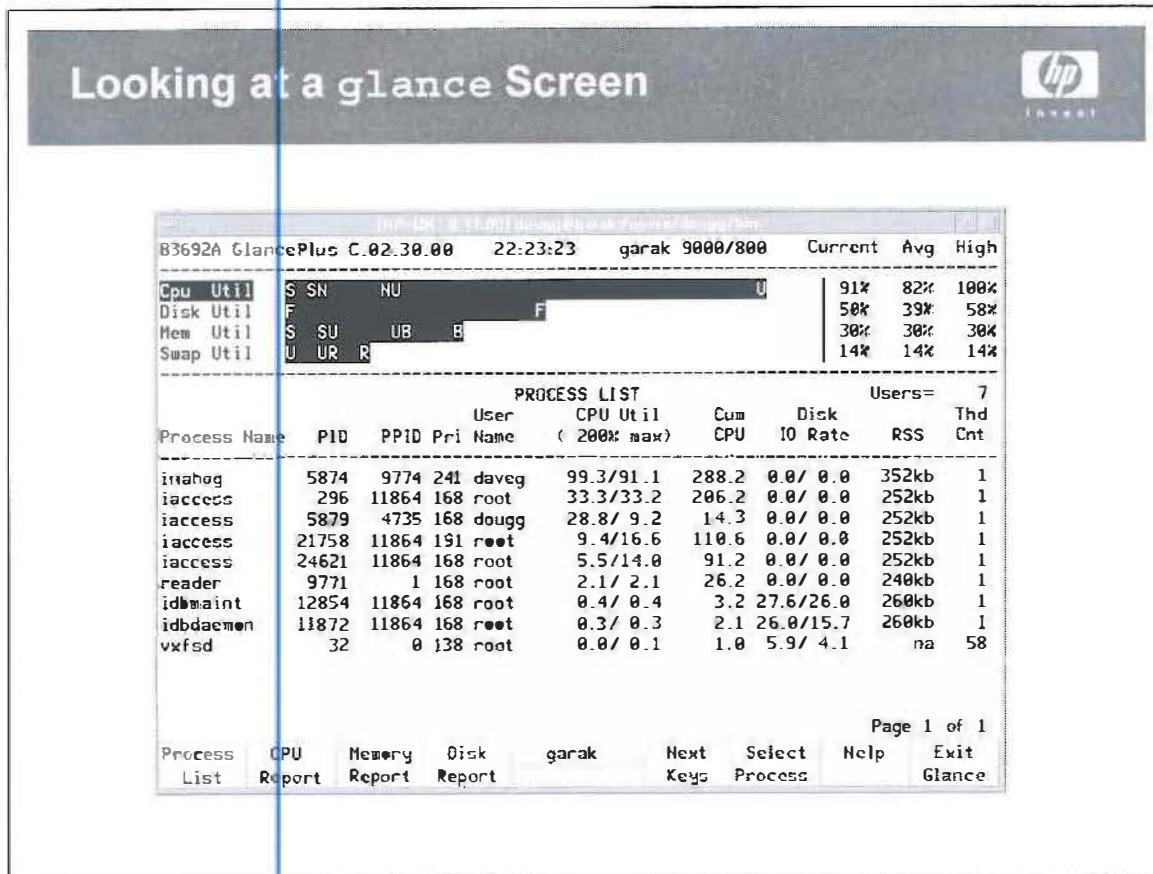
Secondary Level Screen Hot Keys

Hot Key	Screen Displayed/Description
S	Select a NFS system/Disk/Application/Trans/Thread
s	Select a single process
F	Process Open Files
L	Process System Calls
M	Process Memory Regions
R	Process Resources
W	Process Wait States

Miscellaneous Screen Hot Keys

Hot Key	Screen Displayed/Description
b	Scroll page backward
f	Scroll page forward
h	Online HELP
j	Adjust refresh interval
o	Adjust process threshold
p	Print toggle (start/stop auto-printing)
e/q	Quit GlancePlus
r	Refresh the current screen
y	Renice a process
z	Reset statistics to zero
>	Display next logical screen
<	Display previous logical screen
!	Invoke a shell

3-5. SLIDE: Looking at a glance Screen



Student Notes

Above is an example of an easy and common performance problem — a runaway looping process.

Why is the global CPU utilization < 100%, although the sum of the individual process CPU utilizations > 100 %?

Hint: Is this a UP or MP system?

Also note that / (slashes) are used in glance reports to separate current metric values from cumulative averages.

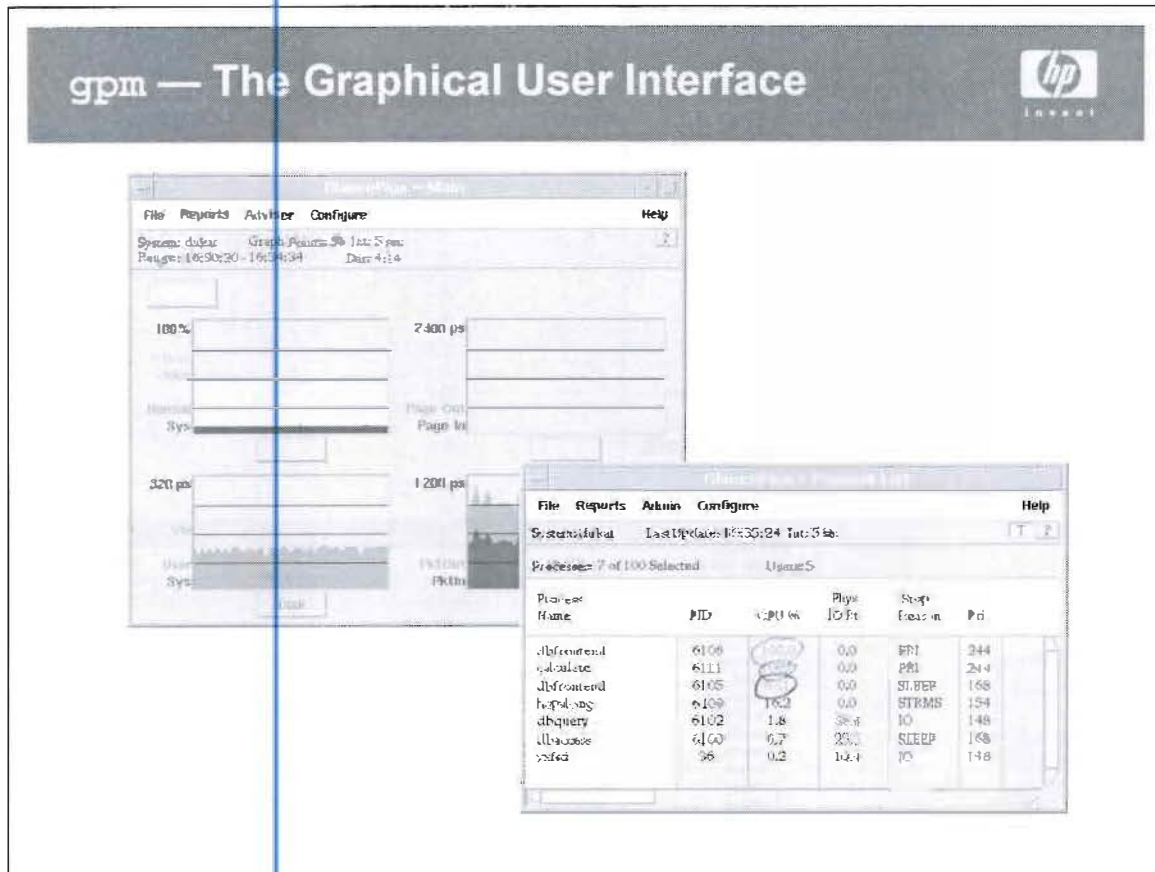
NOTE: For the record there were two CPUs on this system.

Module 3
GlancePlus

On a three-way multiprocessor system with two processes in the same application looping, each process can use nearly 100% of each of 2 CPU's. Over a 10-second interval, each uses nearly 10 seconds of CPU time, so the application used nearly 20 seconds of CPU time in 10 seconds of elapsed time. Process CPU utilization is 100% for each of the 2 looping processes, but global CPU utilization would be 66%.

On HP-UX 11.0, processes can have multiple threads, each of which can consume CPU time independently of the others. On a four-way MP system, with one process that has three threads looping, the process as a total uses 300% of the CPU. The application and global CPU utilization would report the CPU utilization at 75%.

3-6. SLIDE: gpm — The Graphical User Interface



Student Notes

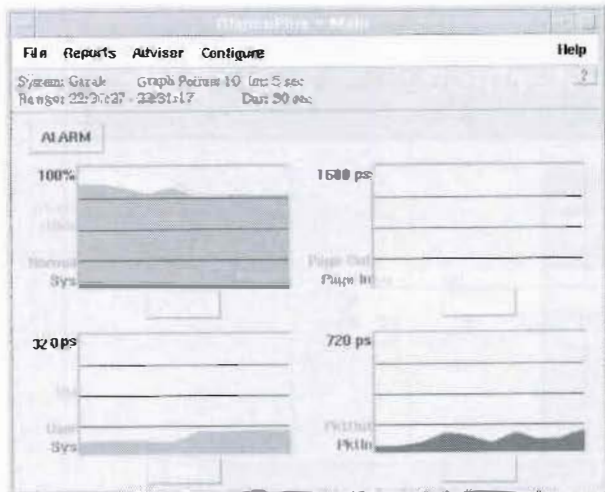
gpm presents the same metrics as character-mode glance in graphical form. Significant global metrics, as well as bottleneck adviser symptom status and alarms are shown in the main window. The process list, as well as other reports, is available via menu selections.

The process list is very customizable (and customizations are preserved) with filters, sorting, highlights, chosen metrics, and column rearrangement.

The online User's Guide is very useful. The  button on every window is a shortcut into the on-item help, which is useful especially for metric definitions.

Module 3 GlancePlus

This is another screen shot of the **gpm** interface.



GlancePlus - Process List							
File Reports Advisor Configure Help							
System: Garak Last Update: 22:31:43 Int: 5 sec							
Processes: 9 of 105 Selected Users: 7							
Process Name	PID	CPU %	Phys MB	Stop Reason	Pri	User Name	Thread Count
initlog	5874	0.0	0.0	PRI	234	daveg	1
lancees	296	26.9	0.0	SLEEP	168	root	1
lancees	21758	11.0	0.0	SLEEP	168	root	1
lancees	24621	5.5	0.0	SLEEP	168	root	1
reader	9771	2.3	0.0	PRI	170	root	1
lancees	5870	1.9	0.0	SLEEP	168	daveg	1
lancees	11872	0.4	28.2	SLEEP	168	root	1
lancees	12854	0.4	27.8	IO	148	root	1
vertd	33	0.0	4.4	SLEEP	138	root	58

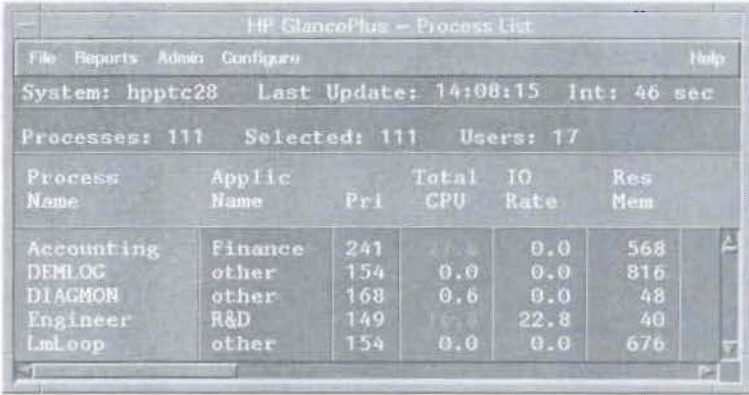
Note the icon reflecting an adviser alarm.

3-7. SLIDE: Process Information

Process Information


Process Information

- Detailed data on each active process
- CPU data
- Disk I/O data
- Memory Use
- Wait Reasons
- Open Files



Process Features

- Access via Main Reports selection Process List
- Each Process has:
 - Process Resources
 - Open Files

Student Notes

The Process Information screen in **gpm** presents the user with detailed information on each active process (including CPU utilization, disk I/O data, memory usage, wait state reasons, open () file information, and so on).

This screen also allows the user to select a specific process and "drill down" to greater detail via the Reports selection menu.

Resource Diagnostic Monitoring

GlancePlus provides an abundant set of performance metrics to help analyze the current system. Careful thought and consideration have been given to ensure that the proper metrics are displayed. The product with its Motif GUI offers a way to efficiently display performance information, without overloading the customer with screen after screen of detailed data.

Customizable GUI

GlancePlus uses the power of Motif and its industry-leading approach to display technology, to provide the user with a powerful graphical user interface that can be customized to fit your needs. Fonts, color, window size and more are configuration options. Additional configuration choices are available in "list" windows to allow easy manipulation of column tabular data for display and sort uses.

The **gpm Process List** and **GlancePlus - Main** screen provide a pull-down menu to access the numerous, detailed **Report** screens. These reports allow a logical approach to the extensive amount of system resources and process specific data.

▼ Resource History Window

▼ CPU Info

▼ Memory Info

▼ Disk Info

▼ Network Info

▼ System Info

▼ Global Info

▼ Swap Space

▼ Wait States

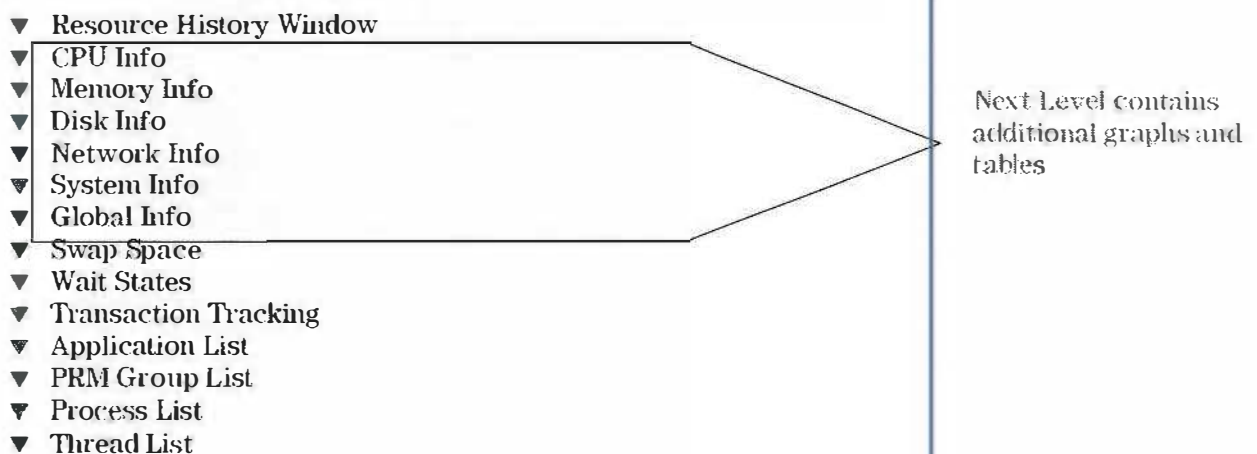
▼ Transaction Tracking

▼ Application List

▼ PRM Group List

▼ Process List

▼ Thread List

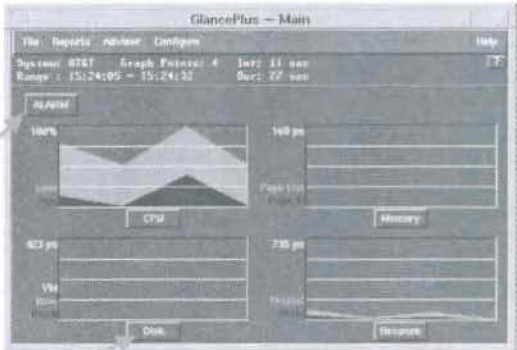


Next Level contains
additional graphs and
tables

3-8. SLIDE: Adviser Components

Adviser Components


- Adviser Windows
 - Symptom History
 - Symptom Status/Snapshot
 - Alarm History
 - Adviser Syntax
- Button Label Colors
 - Alarm Button for Alarm Statements
 - Graph Buttons for Symptom Statements
- Icon Border Color (in OpenView)
 - Changes to Red or Yellow on Alarms



Student Notes

GlancePlus supports performance alarms and a rules-based adviser to help automate the interpretation of performance data. The alarm rules can be customized by the user to reflect local system characteristics.

Note: Both interfaces will report alarms, and the same syntax is used for alarms in **glance** and **gpm**.

Alarms are configured through the `/var/opt/perf/advisor.syntax` file.

3-9. SLIDE: adviser Bottleneck Syntax Example

→ gesture de alarm

adviser Bottleneck Syntax Example



```
# The following symptoms are used by the default Alarm Window
# Bottleneck alarms. They are re-evaluated every interval and
# the probabilities are summed. These summed probabilities are
# checked by the bottleneck alarms. The buttons on the gpm
# main window will turn yellow when a probability exceeds 50%
# for an interval, and red when a probability exceeds 90% for
# an interval. You may edit these rules to suit your environment:

symptom CPU_Bottleneck type=CPU
rule GBL_CPU_TOTAL_UTIL > 75 prob 25
rule GBL_CPU_TOTAL_UTIL > 85 prob 25
rule GBL_CPU_TOTAL_UTIL > 90 prob 25
rule GBL_PAI_QUEUES > 3 prob 25

alarm CPU_Bottleneck > 50 for 2 minutes
start
  if CPU_Bottleneck > 90 then
    red alert "CPU Bottleneck probability= ", CPU_Bottleneck, "%"
  else
    yellow alert "CPU Bottleneck probability= ", CPU_Bottleneck, "%"
  repeat every 10 minutes
  if CPU_Bottleneck > 90 then
    red alert "CPU Bottleneck probability= ", CPU_Bottleneck, "%"
  else
    yellow alert "CPU Bottleneck probability= ", CPU_Bottleneck, "%"
end
reset alert "End of CPU Bottleneck Alert"
```

Student Notes

The bottleneck alarms are a little complex. The CPU bottleneck symptom definition and corresponding alarm is shown.

Just because a resource is fully utilized doesn't mean that it is a bottleneck. It is only a bottleneck if there is activity that is hindered waiting for that resource. Therefore, utilization alone is not a good bottleneck indicator. Both utilization and queue lengths are combined to define the symptom probability.

Some of the key metrics for performance analysis are the ones we use in the default syntax to define bottleneck alarms.

extra mwa stop / mwa start

3-10. SLIDE: The parm File

The parm File


```
application =
user =
file =
priority =
group =
```

! → parm file application definitions are used by both GlancePlus and MeasureWare.

→ A .parm in a user's \$HOME directory will override the system parm file.

application = and the associated parameters defines the logical groupings used to define each application on the machine.

Examples:

```
application=Real Time
priority=0-127
```

```
application=Prog Dev Group 1
file=vi,xdb,abb,ld,lint
user=bill,debbie
```

```
application=Prog Dev Group 2
file=vi,xdb,abb,ld,lint
user=t,ed,reb,ec,ca,ss
```

```
application=Compilers
file=cc,ecom,pc,pascomp
```

Student Notes

By now you are starting to see the range and scope of the performance metric data that glance and gpm display. While this is invaluable when it comes to understanding the behavior of a single process, many times what we really need is to evaluate and baseline the performance of an entire application suite. This could be achieved by adding up the individual metrics of all processes within the application suite, but this could be a daunting task for all but the simplest of applications.

Through the use of the configuration file `/var/opt/perf/parm`, glance and gpm can help to collect all the metrics from the individual processes within an application suite and present the information in a concise manner for your review.

One challenge is in the definition of what constitutes an application. To address this issue, the parm file has several different methods for describing which processes belong to which application definition.

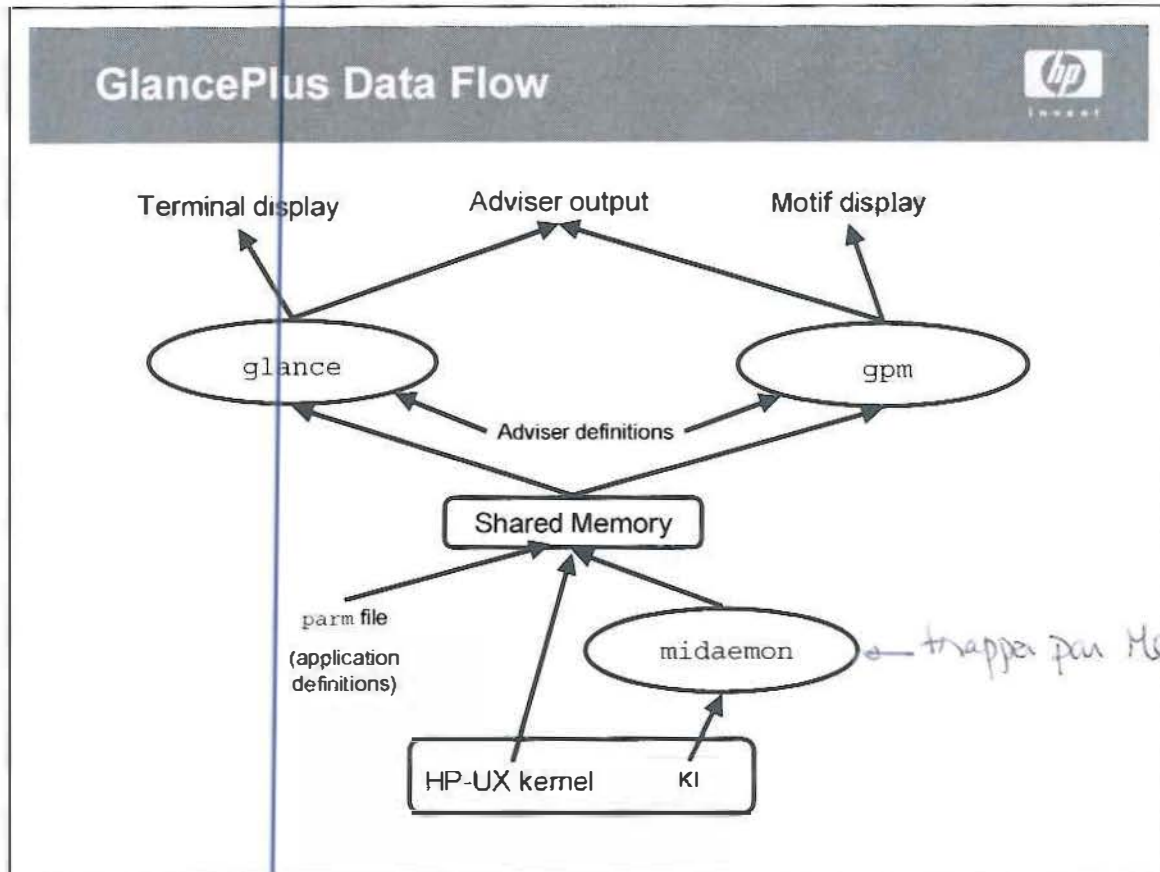
Application member processes can be defined by their UID, the front-store file from which they were fork()'d, the priority at which they execute, their GID, or any combination of the above. This provides a very versatile framework for application profiling.

NOTE: glance and gpm share the same application definitions (via the parm configuration file) as mwa.

```
# /var/opt/perf/parm for host system "garat"
id = garat
# Parameters for what data classes scopeux will log:
log global application process dev=disk,lvm transaction
# Parameters to control maximum size of scopeux logfiles:
size global=10, application=5, process=2, device=1,
transaction=1.5
# Thresholds which determine what process data scopeux will log:
threshold cpu = 1, disk = 1, nonew, nokilled
# Web server:
application = WWW
user = www
or
file = httpd
# Untrustworthy users:
application = HighRisk
user = fred,barney,root
```

The order in which applications are defined is very important. Once a process meets the definition of an application, its data will be contributed to that application's metrics. Care must be taken to assure that ambiguity is avoided in the definition of applications.

3-11. SLIDE: GlancePlus Data Flow



Student Notes

Without going into a lot of detail, note that both interfaces share a common instrumentation source and common application definitions.

Instrumentation comes partly from interfaces also accessed by standard UNIX utilities such as `vmstat`, and partly from special HP-UX KI trace-based instrumentation.

There is no generally available API to these interfaces. They are written specifically for use by GlancePlus and MeasureWare/OVPA.

Significant Directories

<code>/opt/perf</code>	Product files from installation media
<code>/opt/perf/bin</code>	Executables
<code>/opt/perf/ReleaseNotes</code>	Release Notes
<code>/opt/perf/examples</code>	Supplementary configuration examples
<code>/opt/perf/paperdocs</code>	Electronic versions of documentation
<code>/var/opt/perf</code>	Product and configuration files created during and after installation

Always check ReleaseNotes for version-specific information.

(New for C.02.30 and later releases: example configuration files)

Config files come from `/opt/perf/newconfig` if they don't already exist under `/var/opt/perf`.

Compare new default `parm` file with that on your system if you are updating from a previous release.

The directory `/var/opt/perf` contains the status and data files.

3-12. SLIDE: Key GlancePlus Usage Tips

Key GlancePlus Usage Tips



- Use it for “What’s going on right now.”
- The gpm online help is very useful — especially on item help.
- Drill down from higher level reports to more detailed resource reports.
- Understand what the adviser is telling you.
- Sort, filter, and choose metrics in gpm; especially the Process List.
- In character-mode glance use:
 - ? screen to navigate
 - h for help
 - o screen for setting thresholds and process list sorting
- Edit the adviser alarms to be right for you.
- Adjust update interval to control CPU overhead.
- Process details including thread lists, wait states, memory regions, open files, and system call reports can be used to impress your programming staff ! 8^)

Student Notes

3-13. SLIDE: Global, Application, and Process Data

Global, Application, and Process Data



- Global metrics reflect system-wide activity (sum of all applications).
- Process metrics reflect specific per-process (including thread) activity.
- Application metrics sum activity for a set of processes. They keep track of activity for all processes, however short-lived, even if they are not reported individually.
- Glance updates all metric values at the same time. MeasureWare summarizes Global, Application, and other class data over 5-minute intervals and summarizes Process data over 1-minute intervals.
- Multiprocessor effects: Global and Application CPU percentages reflect normalization over the number of processors (percentage of availability for entire system). Process and thread-level CPU percentages are not normalized by the number of processors.

Student Notes

It is important to understand the interrelationships among metric classes.

3-14. SLIDE: Can't Solve What's Not a Problem

Can't Solve What's Not a Problem!



- A looping process by itself is not a problem.
- Know what's "normal" for your environment.
- Keep historical performance data for reference.
- Measure response times.
- Use the tools to find out what is affecting performance.
- Isolate bottlenecks and address them when there is a problem.
- When tuning, make only one change at a time and then measure its effect.
- Document everything you do!
- Optimize your time resource: don't fix what isn't broken; sometimes more hardware is the cheapest answer; set yourself up to react quicker next time.

Student Notes

One of the hardest skills is to determine what to measure and how to interpret its significance. After all, if the user's response time is satisfactory, then oftentimes there is "no problem" even if an operation metric is higher than normal.

3-15. SLIDE: Metrics: "No Answers without Data"

Metrics: "No Answers without Data"



- Rate and utilization metrics are more useful than counts and times, because they are independent of the collection interval.
- Cumulative metrics measure over the total duration of collection.
- Most metrics are broken down into subsets by type. Work from the top down.
- Blocked states reflect individual process or thread wait reasons. Global queue metrics are derived from process blocked states.
- CPU is a "symmetric" resource. Scheduler will balance load on the multiprocessor, whereas disks and network interface activity depend on where data is located.
- Memory utilization is not as important as paging activity and buffer cache sizing.

Student Notes

CPU utilization and disk I/O rates compare well on different summarization intervals, whereas CPU time and I/O counts are always larger when the collection interval grows.

Examples of breakdowns: Global disk I/O rate is a sum of the **BYDSK_** metrics, each class in turn breaks down activity between reads and writes and file system versus raw and system access. For disk bottlenecks, it is often useful to correlate between DSK, FS, and LV classes.

Memory utilization is frequently nearly 100% with dynamic buffer cache. If page outs occur or while in raw disk access environments, shrink the buffer cache to avoid paging.

Programmers frequently don't know they can view specific system-call metrics, as well as memory region and open file information on a per-process basis.

3-16. SLIDE: Summary

Summary



- Don't try to understand all the capabilities and extensions to the tools, just the ones of most use to you.
- Start with developing an understanding of what is "normal" on your systems.
- Refine and develop alarms customized for your environment.
- Work from examples in documentation, gpm online help, config files, and example directories.

Student Notes

Remember that performance tuning is an art, and the following two rules apply to most engagements:

Rule #1: When answering a question about computer system performance, the initial answer is always, "It depends."

Rule #2: Performance tuning always involves a trade-off.

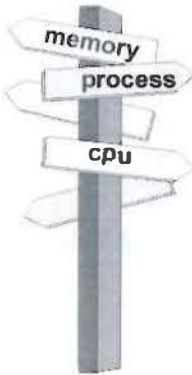


Suggested reading: *HP-UX Tuning and Performance* by Robert F. Sauer and Peter S. Weygant, available through the Hewlett-Packard Professional Books, Prentice Hall Press (ISBN 0-13-102716-6)

3-17. SLIDE: HP GlancePlus Guided Tour

HP GlancePlus Guided Tour





Topics

- Main Window
- CPU Bottlenecks
- Memory Bottlenecks
- Configuration Information
- Alarm and Symptoms

Student Notes

To take the guided tour of GlancePlus, run the **gpm** GUI and select **Help** on the menu bar. Next, select the **Guided Tour** option.

This will introduce you to the product. It features captured “windows” of the actual product, with annotations to help point out the important features of certain screens or windows.

Quick Tip: **gpm** provides an excellent online Help system. Click the right mouse button for the On-Item Help feature. For help in **glance**, press the **h** key.

3-18. LAB: gpm and glance Walk-Through

Directions

The following lab is intended to familiarize the student with gpm and glance. To achieve this result, the lab will “walk the student through” a number of windows and tasks in both the ASCII and X-Windows versions of gpm and glance.

The Graphical Version — GlancePlus

1. **Log in.** If you have not already done so, please log into the system with the user name and password provided by your instructor.
2. **Start GlancePlus.** From a terminal window, invoke GlancePlus by entering `gpm`.

```
# gpm
```

In a few seconds `gpm` will come up. The first thing will be a license notification informing you that you are starting a trial version of GlancePlus, along with ordering and technical support information.

On the `gpm` Main screen, you will see four graphs for CPU, Memory, Disk, and Networking. By default, the graphs are in the resource history format. This means that for each interval (configurable) there will be a data point on the graph, up to the maximum number of intervals (also configurable).

3. **Interval Customizations.** Click on **Configure** in the menu bar, and select **Measurement**. Set the sample interval to 10 seconds and the number of graph points to 50. This will allow you to see up to 500 seconds of system history. Click on **OK**.

NOTE: This setting will be saved for you in your home directory in a file called `$HOME/.gpmhp-system_name`. This means that all GlancePlus users will have their customizations saved.

Start a program from another window:

```
# cd /home/h4262/cpu/lab1;  
# ./RUN &
```

4. **Main Window.** Below each graph within the GlancePlus Main window, you will find a button. These buttons display the status color of adviser symptoms. This is a powerful feature of GlancePlus that we will investigate later. Clicking on one of these buttons displays details of that particular graph.

To view the advisor symptoms from the main window, select:

```
Adviser -> Edit Adviser Syntax
```

This will display the definitions of the current symptoms being monitored by GlancePlus. Close the **Edit Adviser Syntax** window.

Module 3 GlancePlus

View CPU details:

Click the CPU button.

To view a detailed report regarding the CPU, select:

Reports -> CPU Report

Select:

Reports -> CPU by Processor

This is a useful report, even on a single processor system.

5. **On Line Help.** One method for accessing online help within GlancePlus is to click on the question mark (?) button. The cursor changes to a ?.

Click on the column heading, **NNice CPU %**. This opens a new window describing the **NNice CPU %** column.

View descriptions for other columns, including the **SysCall CPU%**.

When finished viewing online help for columns, click on the question mark one more time. This returns the cursor to normal.

6. **Alarms and Symptoms.** A symptom is some characteristic of a performance problem. GlancePlus comes with predefined symptoms, or the user can define his own.

An alarm is simply a notification that a symptom has been detected.

From the main window, select:

Adviser -> Symptom History

For each defined symptom, a history of that particular symptom is displayed graphically. The duration is dependent on the **glance** history buffers, which are user-definable. Close the window.

Click on the **ALARM** button in the main window. This displays a history of all the alarms that have occurred since GlancePlus was started. Up to 250 alarms can be displayed. Close the window.

7. **Process Details.** Close all windows except for the main window. Select:

Reports -> Process List

This shows the "interesting" processes on the system (interesting in terms of size and/or activity). To customize this listing, select:

Configure -> Choose Metrics

This will display an astonishing number of metrics, which can be chosen for display in this report. This is also a quick way to get an overview of all of the process-related

metrics available in GlancePlus. Note that the familiar ? button is also available from this window.

Use the scroll bar to find the metric `PROC_NICE_PRI`. Select this metric and click on OK. Close this window by clicking on OK.

8. **Customizations.** Most display windows can be customized to sort on any metric, and to arrange the metrics in any user-defined order. To define the sort fields, select

Configure -> Sort Fields

The sort order is determined by the order of the columns. Placing a particular metric into column one makes it the first sort field. If multiple entries have the same value within this field, then the second column is used to determine the order between those entries. If further sorting is needed, then the third column is used, and so forth down the line.

To sort on **Cumulative CPU Percentage**, click on the column heading `CPU % Cum`. The cursor will become a crosshair. Scroll window back to column one, and click on column one. This makes `CPU % Cum` the first sort field.

Arrange the sort order so that `CPU %` is followed by `CPU % Cum`.

Click **Done** when finished. This sort order is automatically saved so that the next time processes are viewed, this will remain the sort order.

In a similar fashion, the order of the columns can also be arranged. To define the column order, select

Configure -> Arrange Columns

Select a column to be moved (for example, `CPU % Cum`). The cursor will become a crosshair. Scroll the window to the location where the column is to be inserted. Click on the column where the column is to be inserted.

Arrange the first four columns to be in the following order: `Process Name`, `CPU %`, `CPU % Cum`, `Res Mem`.

Click **Done** when finished. This display order is automatically saved so that the next time processes are viewed, this will remain the display order.

9. **More Customizations.** It is possible to modify the definition of interesting processes by selecting:

Configure -> Filters

An easy way to limit the processes shown is to *and* all the conditions (the default is to *OR* the conditions). In the **Configure Filters** window, select **AND** logic, then click on OK. A much smaller list of processes should be displayed.

Return to the **Configure Filters** window. Modify the filter definition for `CPU % Cum` as follows:

Change Enable Filter to ☒ N

Change Filter Relation to `>=`

Module 3 GlancePlus

Change Filter Value to 3.0

Change Enable Highlight to ON

Change Highlight Relation to >=

Change Highlight Value to 3.0

Change Highlight Color to any LOUD color

Reset the logic condition make to OR, then click OK. Verify the filter took effect.

10. **Administrative Capabilities.** There are two administrative capabilities with GlancePlus. If working as root, processes in the **Process List** screen can be killed or reniced.

In the **Process List** window, select the **proc8** process. To access the **Admintools**, select:

Admin -> Renice

Use the slider to set the new **nice** value for this process to be +19, then click OK. Note the impact on this process.

Now, select the **proc8** process again. Select:

Admin -> Kill

Click OK, and note the process is no longer present.

11. **Process Details.** Detailed metrics can be obtained on a per process basis. To view process details, go to the **Process List** window and double click on any process.

Much of the details in this report will be explained in the **Process Management** section of the course.

The **Reports** menu provides much valuable information about the process, including the **Files Open** and the **System Calls** being generated.

After surveying the information available through this window, close and return to the **Main** window.

There are many other features available in GlancePlus. There are close to 1000 metrics available with it. Notice that when you iconify the GlancePlus **Main** window, all of the other windows are closed and the GlancePlus active icon is displayed. Alarms and histograms are displayed in this active icon. Exploding this icon will again open up all previously open windows.

12. **Exit GlancePlus.** From the **Main** window, select:

File -> Exit GlancePlus

13. **Glance, the ASCII version.** From a terminal window, which has not been resized, type `glance`.

NOTE: Never run `glance` or `gpm` in the background.

If you are accessing the ASCII version of `glance` from an **X terminal window**, make sure you start up an `hpterm` window to enable full `glance` softkeys. Do not resize the window as ASCII `glance` expects a standard terminal size. . You can make the `hpterm` window longer, but never wider. However, making it longer is frequently of no use.

```
# hpterm &
```

In the new window...

```
# glance
```

Display a list of keyboard functions by typing `?`. This brings up a help screen showing all of the command keystrokes that can be used from the ASCII version of GlancePlus. Explore these to familiarize yourself with the interface.

14. **Display Main Process Screen.** Type `g` to go to the **Main Process Screen**. This lists all *interesting* processes on the system.

Retrieve online help related to this window by typing `h`, which brings up a help menu. Select:

```
Current Screen Metrics
```

Use the cursor keys to select

```
CPU Util
```

NOTE: This metric has two values. Use the online help to distinguish the difference between the two values. Use the space bar or the "Page Down" key to toggle to the next page of help.

Exit the online help `CPU Util` description by typing `e`. Exit the **Screen Summary** topics by typing `e`. From the main Help menu, select:

```
Screen Summaries
```

Use the cursor keys to select

```
Global Bars
```

From this help description, explain what **R**, **S**, **U**, **N**, and **A** mean in the `CPU Util Bar`.

Exit the online help `Global Bar` description by typing `e`. Exit the **Screen Summary** topics by typing `e`. Exit the main Help menu by typing `e`. At any time, you can exit help completely, no matter how deep you are, by pressing the **F8** key.

Module 3
GlancePlus

15. **Modify Interesting Process Definition.** From the main Process List window, (select **g**). View the *interesting* processes. What makes these processes interesting? Type **o** and select **1** (one) to view the process threshold screen.

Cursor down to the **Sort** key field, and indicate to sort the processes by CPU usage.

Before confirming the other options are correct, note that any CPU usage (greater than zero), or any disk I/Os will cause the process to be considered *interesting*.

Run the **KILLIT** command to stop all lab loads.

16. **Glance Reports.** This is the *free* form part of the lab. Spend the rest of your lab time going through the various Glance screens and GlancePlus windows. Use the table below to produce the different performance reports.

Feel free to use this time to ask the instructor "How Do I . . .?" types of questions.

Glance		GlancePlus (gpm)
COMMAND	FUNCTION	"REPORT"
*a	All CPUs Performance Stats	CPU by Processor
b	Back one screen	
*c	CPU Utilization Stats	CPU Report
*d	Disk I/O Stats	Disk Report
e	Exit	
f	Forward one screen	
*g	Global Process Stats	Process List
h	Help	
*i	I/O by Filesystem	I/O by Filesystem
j	Change update interval	
*l	Lan Stats	Network by LAN
*m	Memory Stats	Memory Report
*n	NFS Stats	NFS Report
o	Change Threshold Options	
p	Print current screen	
q	Quit	
r	Redraw screen	
*s	Single process information	Process List, double click process
*t	OS Table Utilization	System Table Report
*u	Disk Queue Length	Disk Report, double-click disk
*v	Logical Volume Mgr Stats	I/O by Logical Volume
*w	Swap Stats	Swap Detail
y	Renice process	Administrative Capabilities
z	Zero all Stats	
!	Shell escape	
?	Help with options	
<CR>	Update screen data	

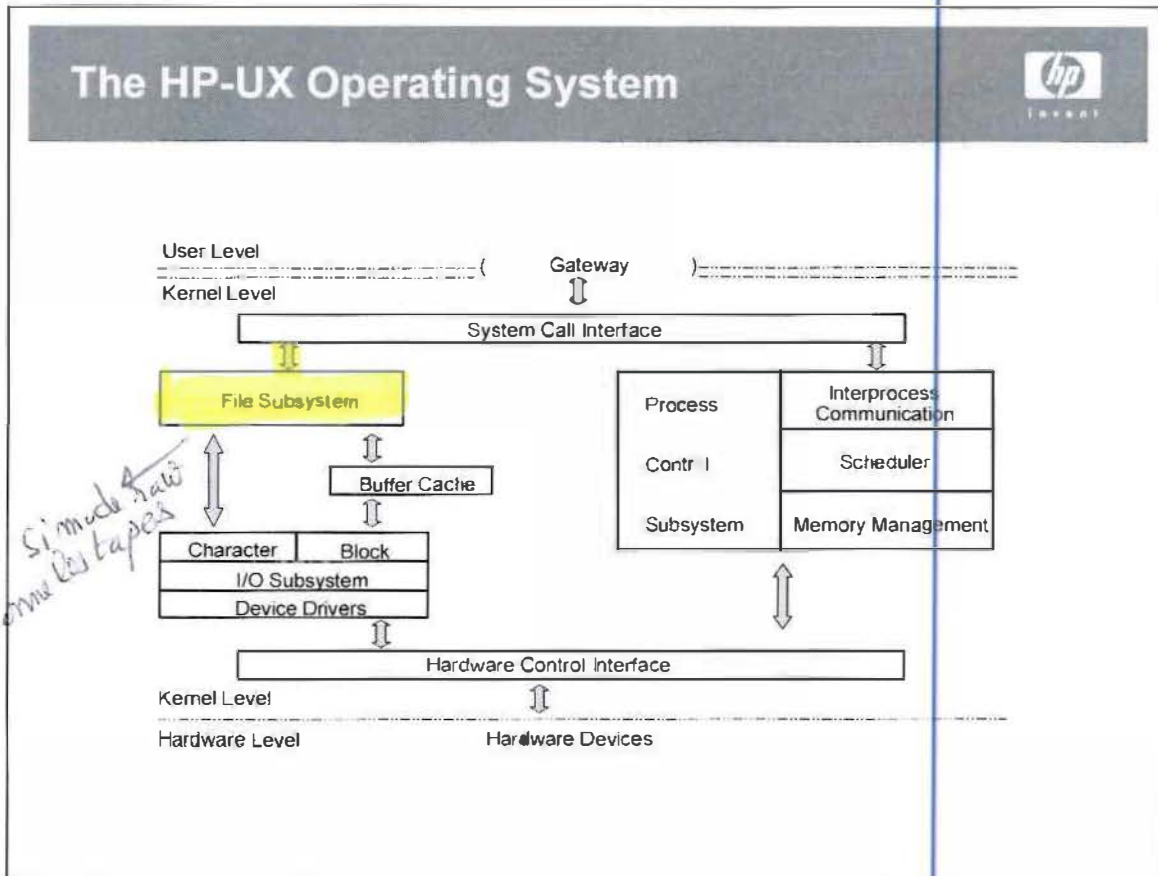
Module 4 — Process Management

Objectives

Upon completion of this module, you will be able to do the following:

- Describe the components of a process.
- Describe how a process executes, and identify its process states.
- Describe the CPU scheduler.
- Describe a context switch and the circumstances under which context switching occurs.
- Describe in general, the HP-UX priority queues.

4-1. SLIDE: The HP-UX Operating System



Student Notes

The main purpose of an operating system is to provide an environment where processes can execute. This includes scheduling processes for time on the CPU, managing the memory which is assigned to processes, allowing processes to read data from disk and many other things.

When processes execute within the HP-UX operating system, there are two modes that they can be in: User mode and Kernel (system) mode.

User Mode and Kernel Mode

User mode refers to instructions that do not require the assistance of the kernel program in order to execute. These include numeric calculations, string manipulations, looping constructs, and many others. In general, it is good when a process can spend the majority of its time in "user mode", because it implies the CPU is executing instructions that are related to the process, as opposed to instructions related to the kernel.

Kernel mode refers to time spent in the kernel executing instructions on behalf of the process. Processes access the kernel through system calls, often referred to as the System Call Interface. Examples include performing I/O, creating new processes, and expanding data space.

Kernel mode is also used for "background" activities, performed by the kernel on behalf of processes. Examples include page faulting the program's text or data in from disk, initializing and growing a process's data space, paging a portion of the process to swap space, performing file system reads and writes, and many other things. In general, when a process spends too much time in kernel mode, it is considered bad for performance. This is because too much time (overhead) is being spent to manage the environment in which the process executes, and not enough time on executing the actual process itself (which is user mode).

Performance Tools

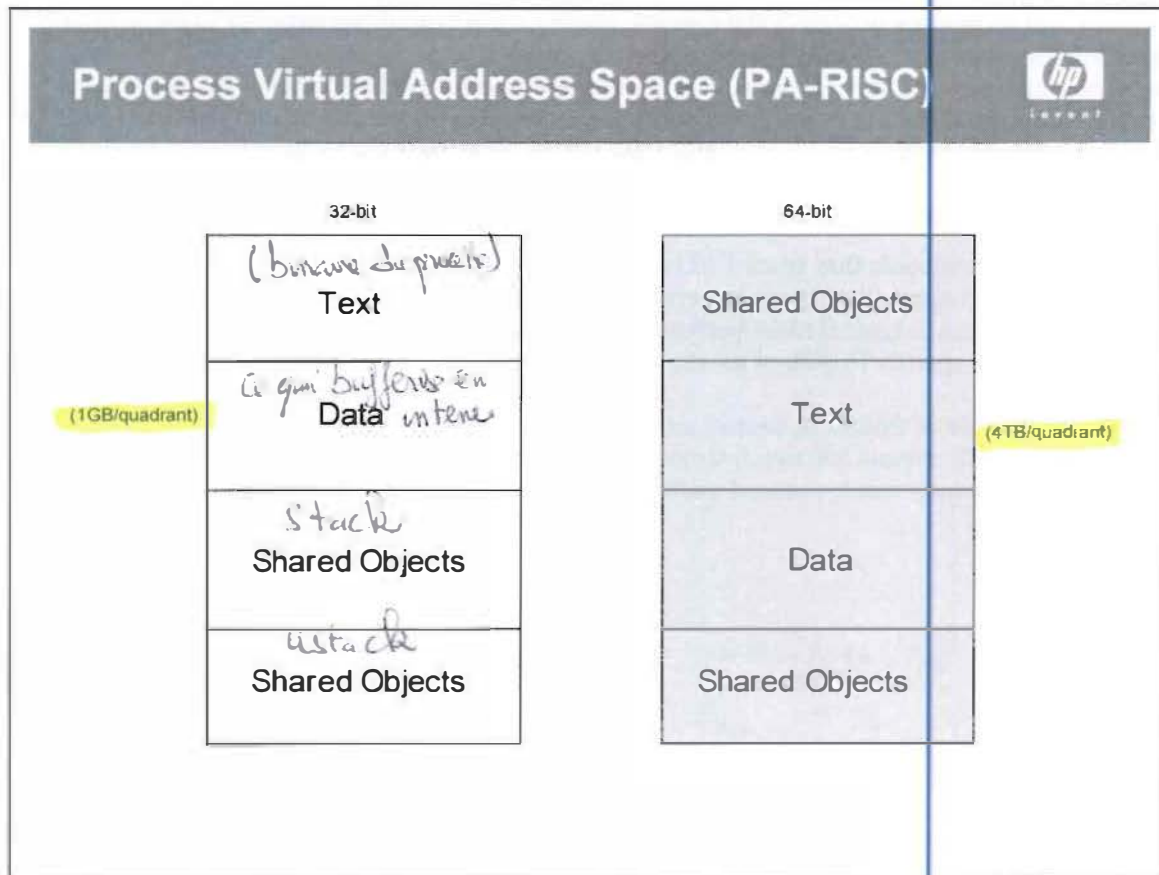
Most all performance tools that track CPU utilization distinguish between time spent by the CPU in *user* mode versus time spent in *kernel* mode. On a good, healthy system with plenty of memory resources, a typical ratio between user mode and kernel mode time is 4:1. This means the process spends 75-80% of its execution in user mode and 20-25% in kernel mode.

Another general rule of thumb is, kernel mode CPU time should not exceed 50%. When this happens, it generally means too much time is being spent managing the system (i.e. memory and swap space management, context switching), and not enough is being spent executing process code.

un user lit un fichier

soit qu'il fait appel au system ou à l'ops (segmentation)
comme oracle

4-2. SLIDE: Virtual Address Process Space (PA-RISC)



Student Notes

Each process views itself as starting at address 0 and ending at the maximum address addressable by 32 or 64 bits. This address space is known as the **Virtual Address Space** for a process.

The virtual address space is a logical addressing scheme used internally by the process to reference related instructions and data variables. The physical memory address locations cannot be used, because a program does not know where in physical memory it will be loaded. In fact, a program could be loaded at different memory locations each time it executes.

The Four Quadrants (32-bit)

Each process segments its virtual address space into four quadrants, with each quadrant containing 1 GB of address space. The first quadrant is reserved for the program's instructions (also known as text). Though an address range of 1 GB is reserved for text, very rarely does the program need all these addresses. Most of the time, only a fraction (often less than 10%) of this space is needed to address the program's text.

pour le passage de 32 bits à 64 bits il faut s'occuper
 1) est ce que le hardware (spec) peut supporter l'indicateur de plus de 4 GB

Module 4 Process Management

The second quadrant holds the programs private data variables. Again, 1 GB of address space is reserved for data variables, and only a fraction of this space is used (in general). Since this quadrant is limited to 1 GB of address space, a maximum global data size of approximately 900 MB is imposed. (In HP-UX, changes were made to allow the global data to use addresses in other quadrants for private data, thereby increasing its maximum size to 3.9 GB.)

The third and fourth quadrants are usually used to address shared memory segments, shared text segments, shared memory-mapped files, and other shared structures, such as the System Call Interface.

64-Bit HP-UX 11.00 Update

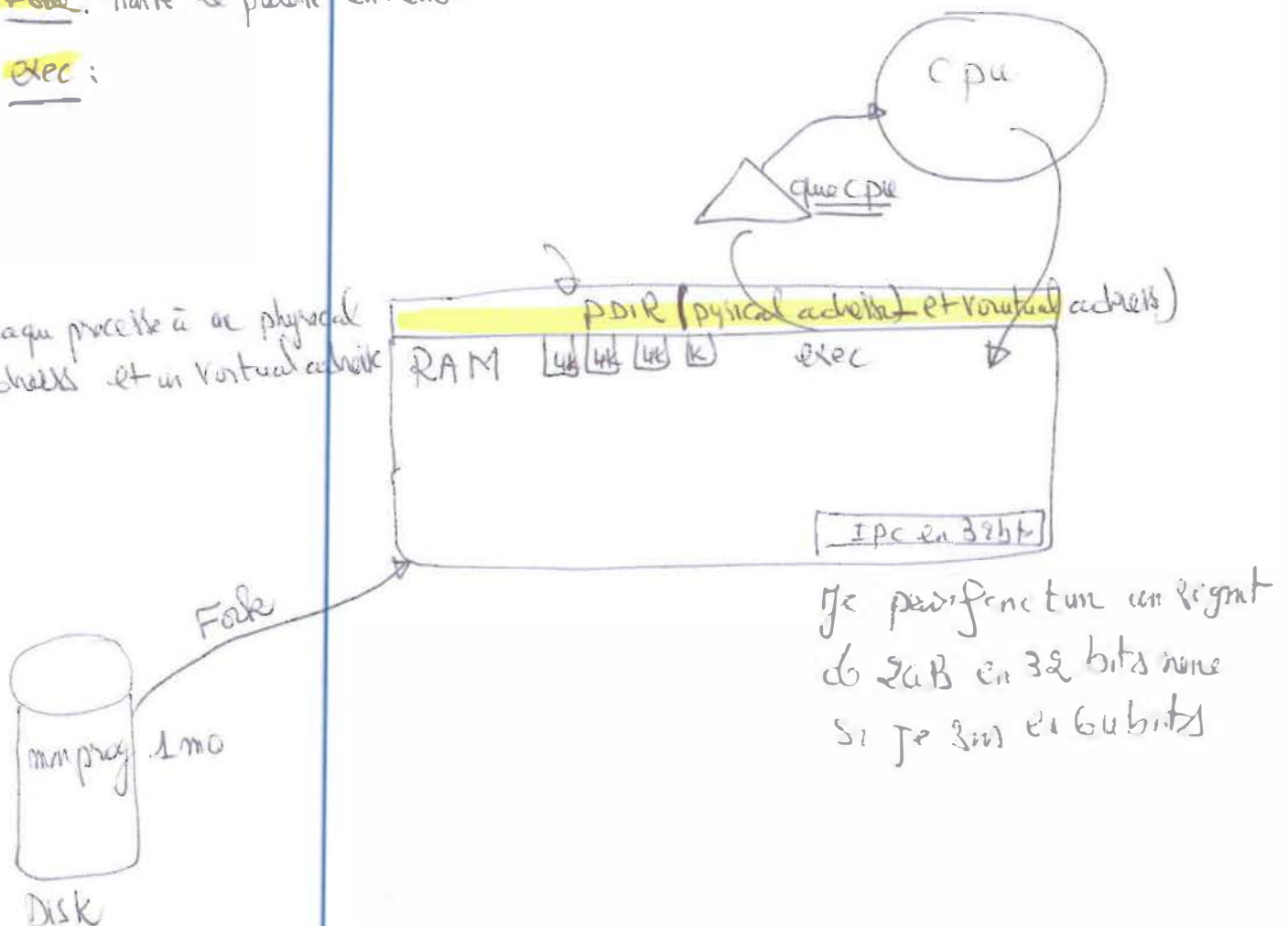
With the introduction of HP-UX 11.00 and its 64-bit operating system, the virtual address space changes dramatically. A 32-bit process running under the 64-bit kernel is given the same space allocations as under a 32-bit kernel. With a 64 bit process, the addressable space increases to 16 Terabytes. This limits each quadrant to 4 TB (for a total of 16 TB of virtual address space), but the capability exists to increase this address space, if necessary, in future releases.

Notice also that the locations of the various components of the process have been shifted among the quadrants.

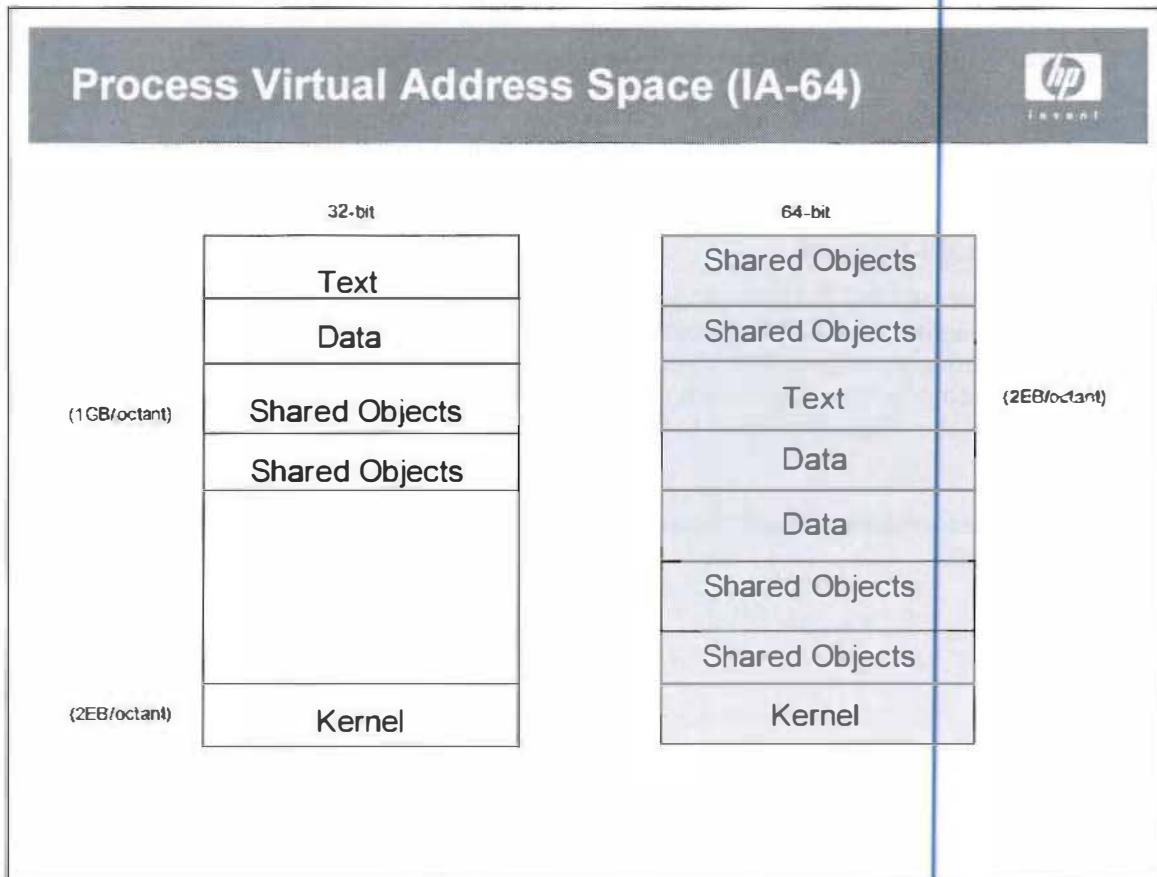
1) code: n'est le programme en mémoire

2) exec:

Chaque processeur a un physical address et un virtual address



4-3. SLIDE: Virtual Address Process Space (IA-64)



Student Notes

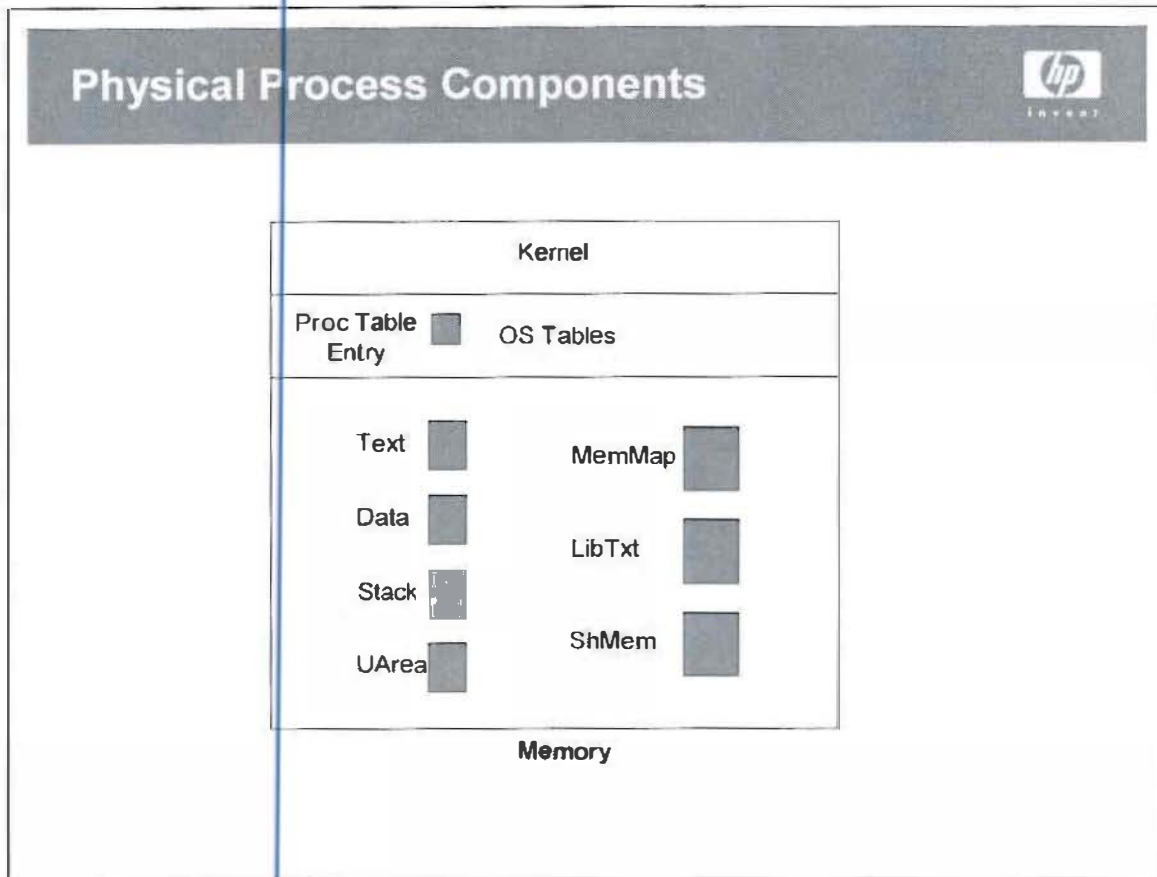
There is no 32-bit kernel running on the IA-64 processor. The virtual address space is always 16 EB in size, although it may not all be used or allocated while a particular process is running. The space is divided into eight equal-sized octants – each octant is 2 EB in size.

When executing a PA-RISC 32-bit process, the first four octants are set up just like the PA-RISC 32-bit virtual address space, using only 1 GB out of each octant to simulate the four original quadrants. The last octant holds the kernel and all of its related structures.

64-Bit Processes

With a 64-bit process, the virtual address space changes dramatically. The first two octants become the equivalent to the first PA-RISC quadrant and hold shared objects. The third octant holds the text. The fourth and fifth octants are reserved for any process private data, and the sixth and seventh octants contain more shared objects. Only the last octant is laid out exactly the same for both 32-bit and 64-bit processes.

4-4. SLIDE: Physical Process Components



Student Notes

Each process executing in memory contains an entry in the kernel's process table. The entry in the proc table then references the locations of the program's four main components: **text**, **data**, **stack**, and **uarea**.

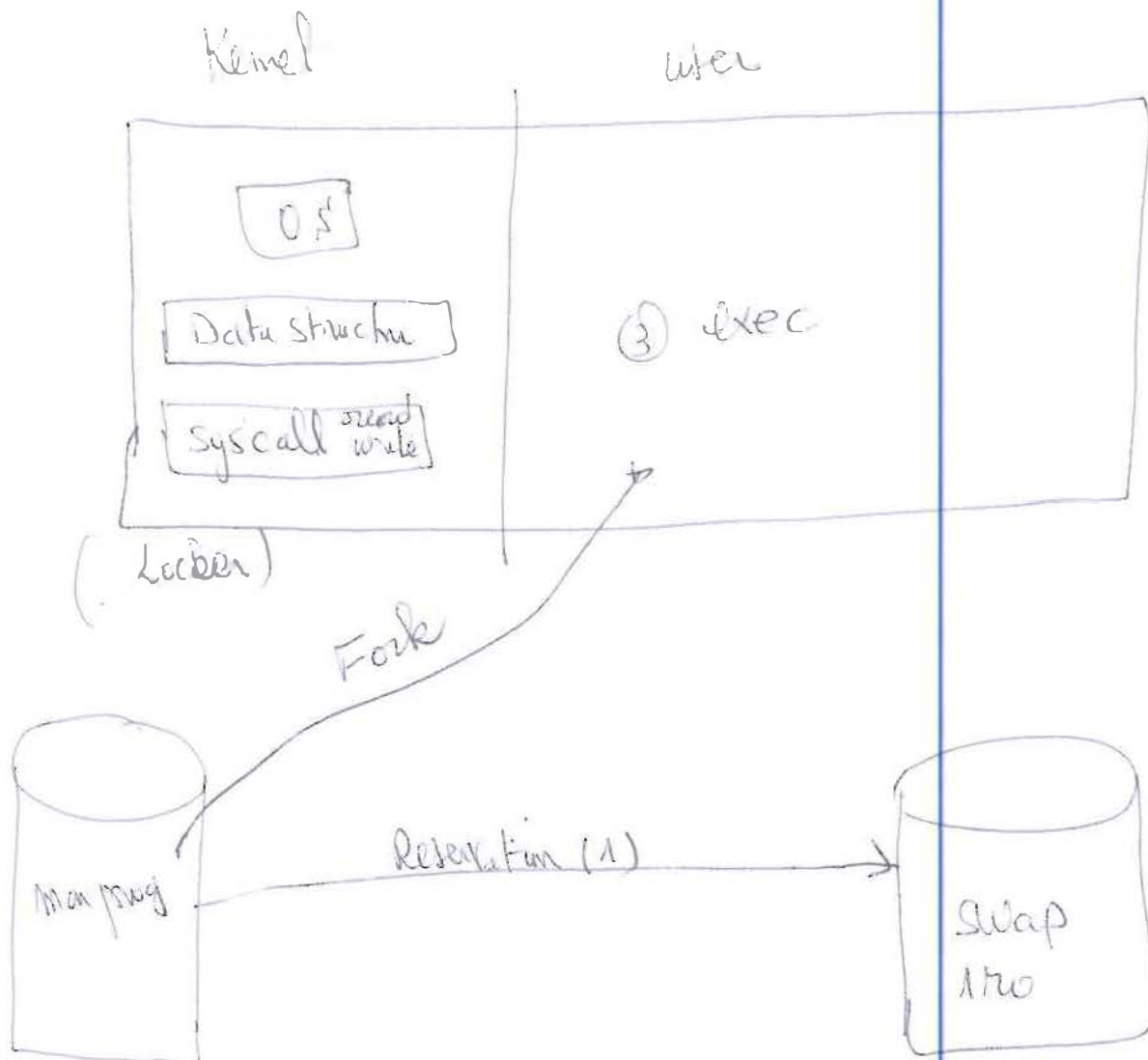
The text segment contains the program's executable code. The data segment contains the programs' global data structures and variables. The stack area contains the programs' local data structures and variables. The uarea is an extension of the proc table entry. In a multi-threaded process, each thread will have its own uarea.

Other components that may or may not be associated with a process are shared libraries, shared memory segments, and memory-mapped files.

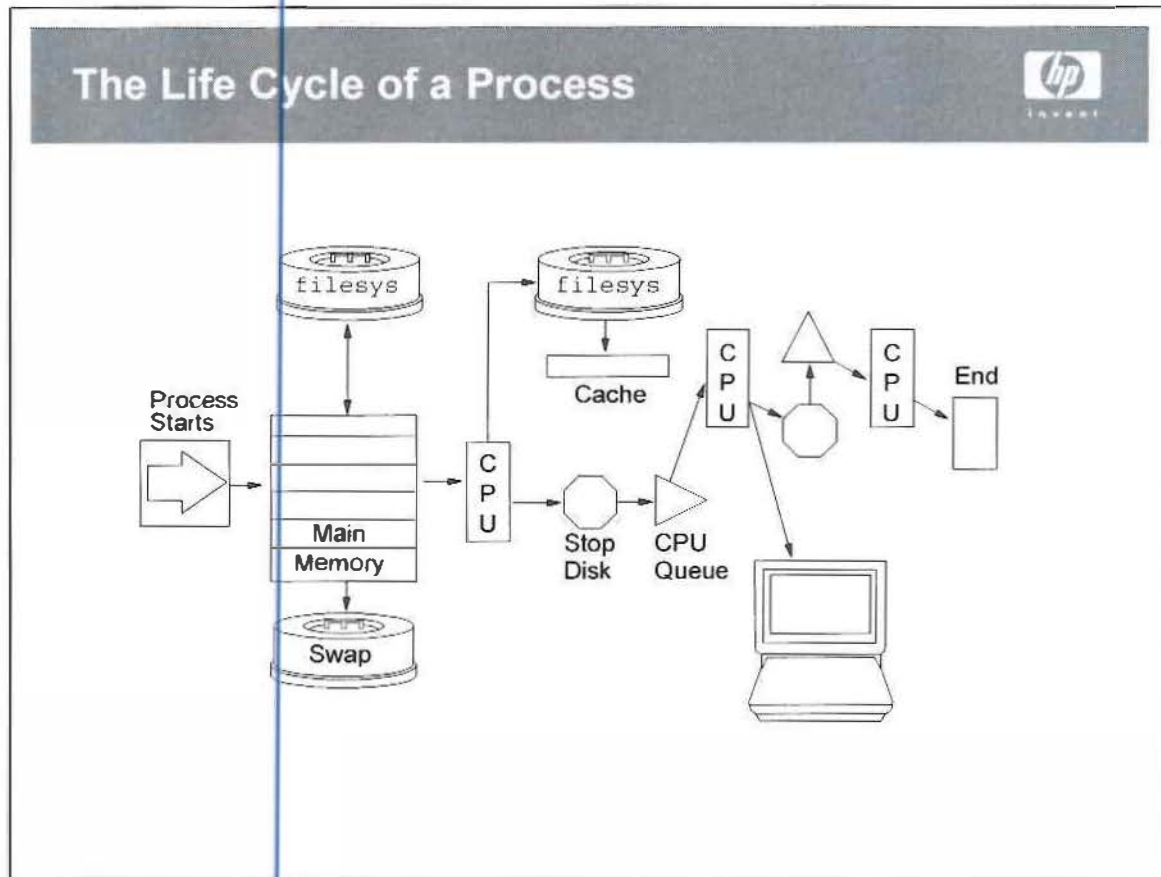
The text and initialized global data segments of the process are taken from the executed program file on disk during process startup. In an attempt to save on startup time, the uninitialized global data segments and the stack area are zero filled, and no pages of a program are loaded at startup. Copying the entire text and data into memory would generate **long startup latency**. This latency problem is avoided in HP-UX by demand paging the program's text and data as needed.

Module 4
Process Management

Using this demand paging approach, the program is loaded into memory in smaller pieces (pages), on an as-needed basis. One page on HP-UX 10.X is equal to a 4-K size. On HP-UX 11.00, the page size is variable (meaning the initial program could page in sizes greater than 4 KB).



4-5. SLIDE: The Life Cycle of a Process



Student Notes

The life cycle of a process can be generalized by the above slide. When a process is born (or starts), its text must be paged in from the file system on disk (on demand) in order to be executed. (Remember, the operating system only pages in a text page when it determines that a process needs a particular page in order to execute.) In addition, space must be reserved on the swap partition for the process in the event it may need to page portions of the data area out to swap. Once the swap space is reserved and the process is initialized, the process can begin executing on the CPU. As the process executes, it often performs actions that require it to wait. These actions include reading data from the disk or the network, waiting for a user to enter a response at a terminal window, or waiting on a shared resource (like semaphores). Once the item, which the process is waiting on, becomes available, the process puts itself in the CPU run queue so it can begin executing again. This is the standard cycle that a process goes through: WAIT for a resource, enter the CPU run queue when the resource is available, execute on the CPU.

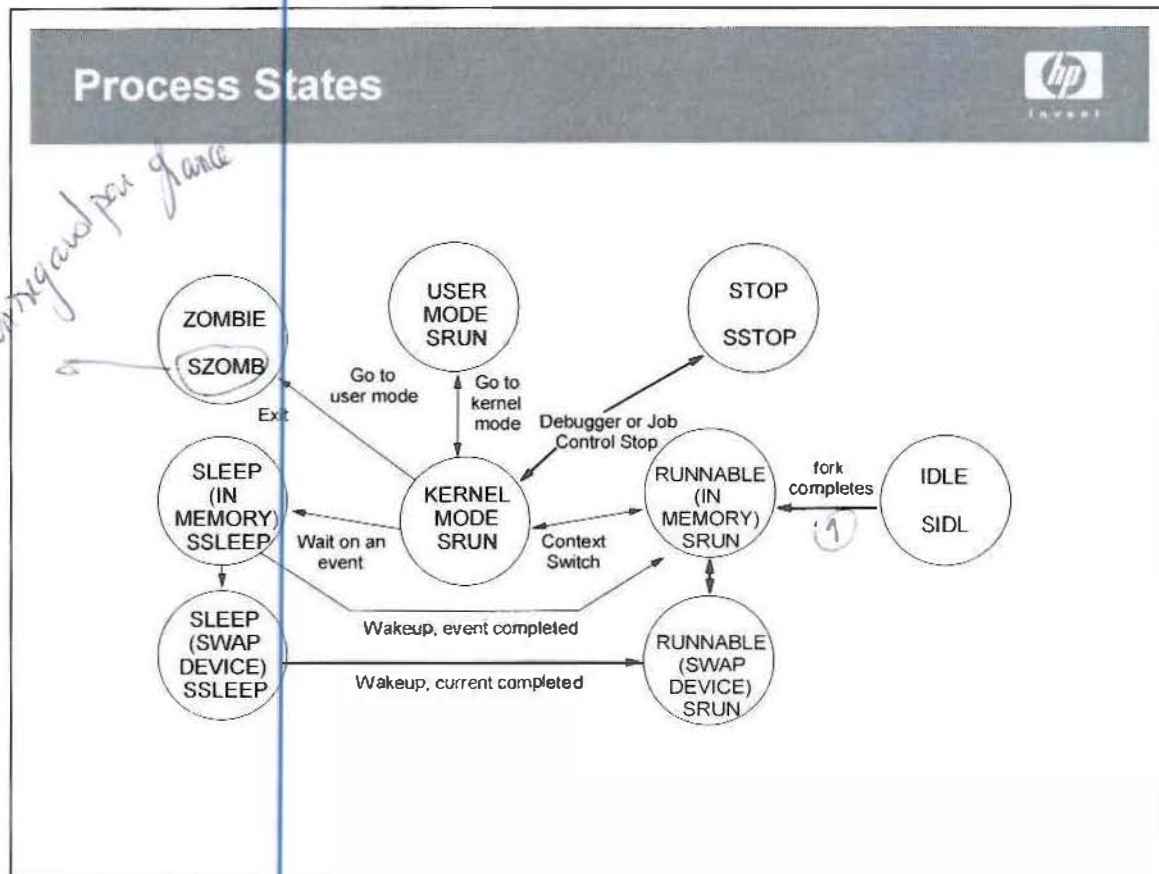
The waiting on a resource is symbolized in the slide as the octagon (or stop sign). The entering of the CPU run queue is symbolized by the triangle, and the execution on the CPU is indicated by the CPU in the rectangle.

Module 4

Process Management

An advantage of the **glance** performance tool is that it displays on a per process basis (or system-wide) the various reasons why a process is blocked or waiting on the CPU.

4-6. SLIDE: Process States



Student Notes

The process table entry contains the process state. This state is logically divided into several categories of information to do the following: scheduling, identification, memory management, synchronization, and resource accounting.

There are five major process states:

SRUN	The process is running or is runnable, in kernel mode or user mode, in memory or on the swap device.
SSLEEP	The process is waiting for an event in memory or on the swap device.
SIDL	The process is being setup via fork.
SZOMB	The process has released all system resources except for the process table entry. This is the final process state.
SSTOP	The process has been stopped by job control or by process tracing and is waiting to continue.

Module 4

Process Management

Most processes, except the currently executing process, are placed in one of three queues within the process table: a run queue, a sleep queue, or a deactivation queue. Processes that are in a runnable state (ready for CPU) are placed on a run queue, processes that are blocked awaiting an event are located on a sleep queue, and processes that are temporarily out of the scheduling mix are placed on a deactivation queue. Deactivated processes typically only occur during a system memory management crisis.

Processes either terminate voluntarily through an `exit` system call or involuntarily as a result of a signal. In either case, process termination causes a status code to be returned to the parent of the terminating process. This termination status is returned to the parent process using a version of the `wait ()` system call.

Within the kernel, a process terminates by calling the `exit ()` routine. The `exit (0)` routine completes the following tasks: cancels any pending timers, releases virtual memory resources, closes open file descriptors, and handles stopped or traced child processes.

Next, the process is taken off the list of active processes and is placed on a list of zombie processes, which is finally changed to being a no process state. The `exit ()` routine continues to record the termination status in the `proc` structure, bundles up the process's accumulated resource usage for accounting purposes, and notifies the deceased process's parent.

If a process in SZOMB state is found, the `wait ()` system call will copy the termination status from the deceased process and then reclaim the associated process structure. The process table entry is taken off the zombie list and returned to the `freeproc` list.

As of IIP-UX 10.10, the concept of a thread was introduced into the kernel. Processes became an environment in which one (or more) threads could execute. Each thread was visible and manageable by the kernel separately. When this occurred, processes were in any of the following states:

SINUSE	The process structure is being used to define one or more threads.
SIDL	The process is being setup via <code>fork</code> .
SZOMB	The process has released all system resources except for the process table entry. This is the final process state.

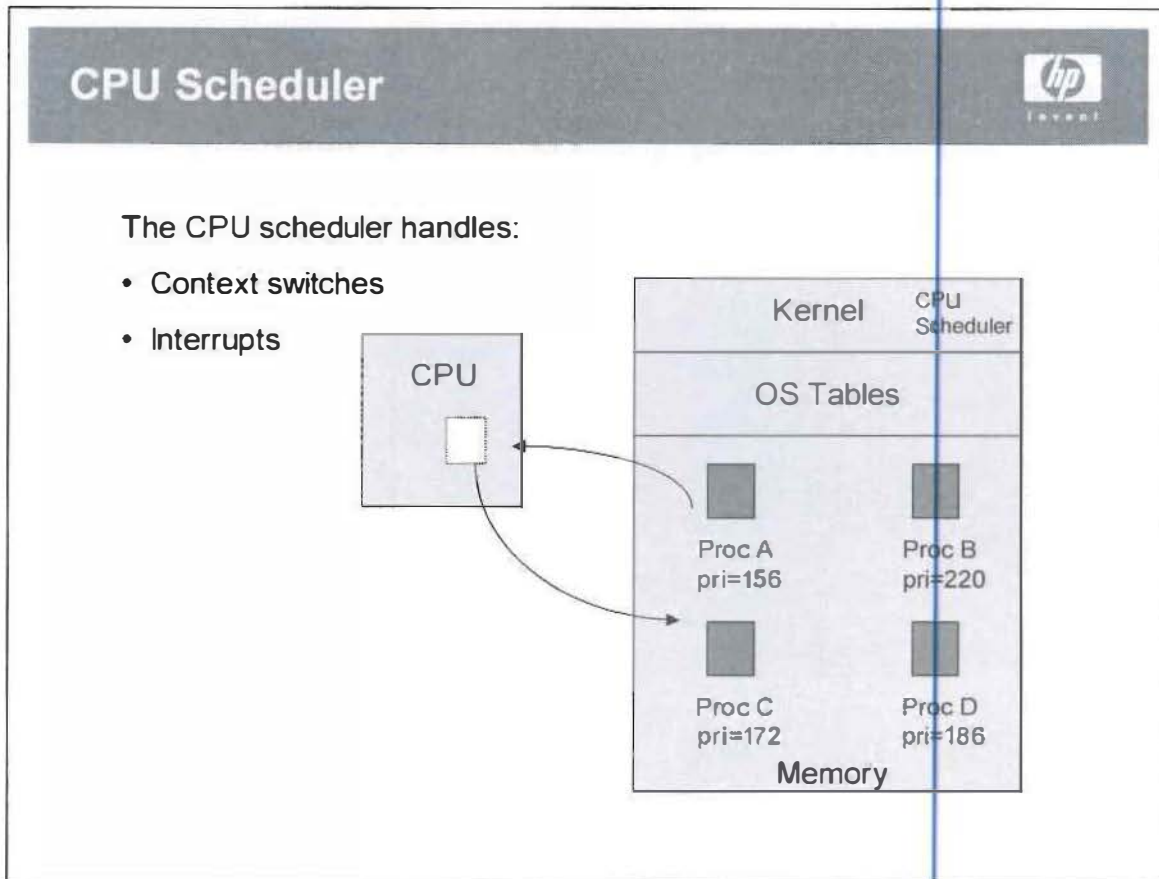
Whereas threads now took on the previous states of the process:

TSRUN	The thread is running or is runnable, in kernel mode or user mode, in memory or on the swap device.
TSSLEEP	The thread is waiting for an event in memory or on the swap device.
TSIDL	The thread is being setup via <code>fork</code> .
TSZOMB	The thread has released all system resources except for the thread table entry. This is the final thread state.

TSSTOP The thread has been stopped by job control or by process tracing and is waiting to continue.

The generic UNIX tools have no awareness of threads and so they continue to report process states and all other metrics from the viewpoint of the process. Only the HP-specific tools (such as **glance**, **gpm**, **PerfView/OVPM**, and **MeasureWare/OVPA**) have the ability to look at individual threads and report their metrics separately from the process. Of course, the vast majority of processes are single-threaded. In those cases, there is no practical difference between the reports of the various tools.

4-7. SLIDE: CPU Scheduler



Student Notes

Once the required data is available in memory, the process waits for the CPU scheduler to assign the process CPU time. CPU scheduling forms the basis for the multitasking, multiuser operating system. By switching the CPU between processes that are waiting for other events, such as I/O, the operating system can function more productively. HP-UX uses a round robin scheduling mechanism.

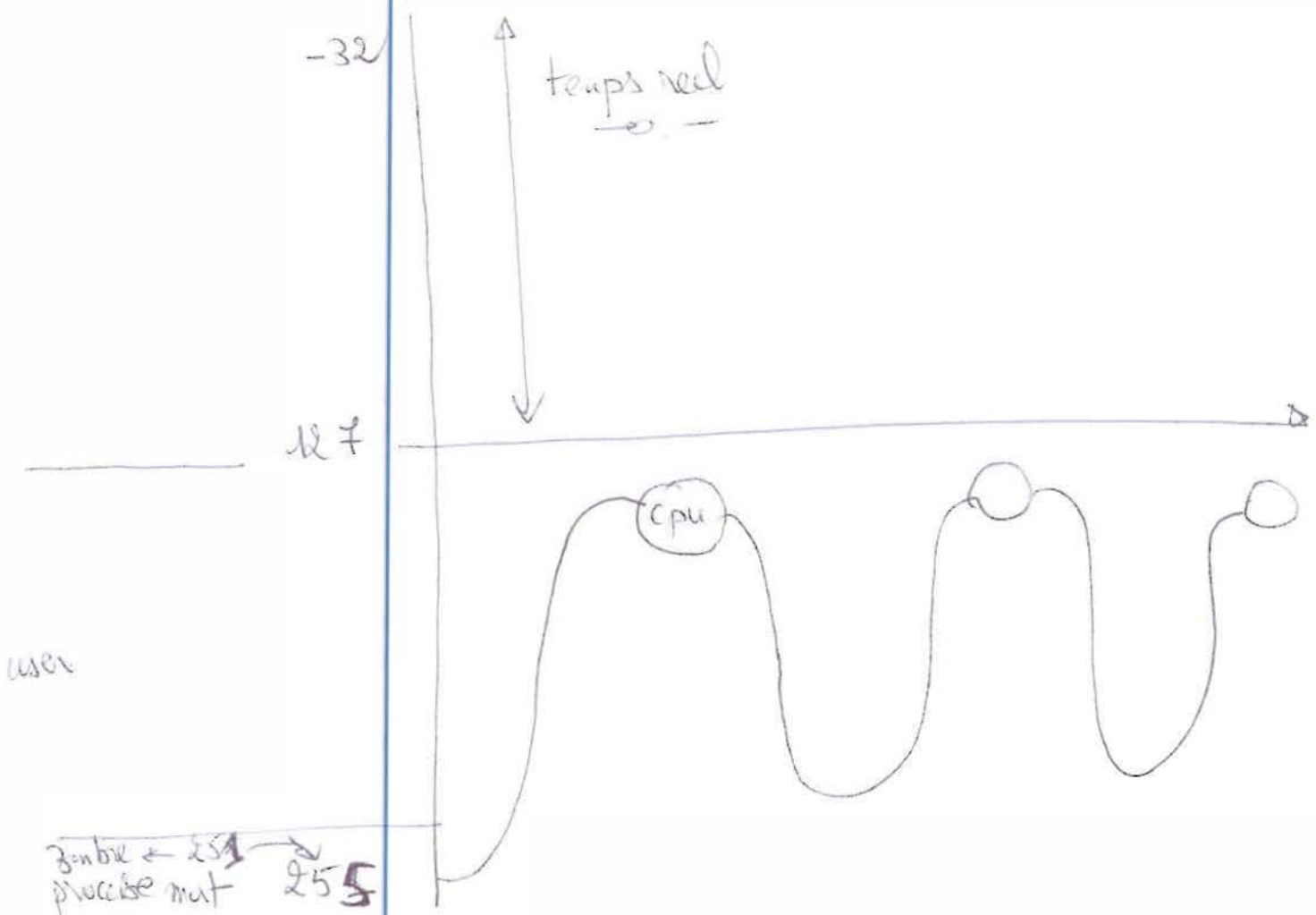
The CPU lets each process run for a preset maximum amount of time, called a quantum or time slice (default = 1/10th second), until the process completes, or is preempted to let another process run. Of course, the process can always voluntarily surrender the CPU before its timeslice expires when it realizes that it cannot continue. The CPU saves the status of the first process in a context and switches to the next process.

When a process is switched out due to its timeslice expiring, it drops to the bottom of the run queue to wait for its next turn. If it is preempted by a stronger priority process, it is placed back onto the front of the run queue. If it voluntarily gives up the CPU, it goes onto one of the sleep queues, until the resource it's waiting for becomes available. When that resource does become available, the process moves the end of the run queue.

As a multitasking system, HP-UX requires some way of changing from process to process. It does this by interrupting the CPU to shift to the kernel. The clock interrupt handler is the system software that processes clock interrupts. It performs several functions related to CPU usage including gathering system and accounting statistics and signaling a context switch. System performance is affected by how rapidly and efficiently these activities occur.

Terms

CPU scheduler	Schedules processes for CPU usage
System clock	Maintains the system timing
Clock Interrupt handler	Executes the clock interrupt code and gathers system accounting statistics
Context switching	Interrupts the currently running process and saves information about the process so that it can begin to run after the interrupt, as if it had never stopped.



4-8. SLIDE: Context Switching

Context Switching



A context switch occurs when

- A timeslice expires (a thread accumulates 10 clock ticks)
(Forced)
- A preemption occurs (a stronger priority thread is runnable)
(Forced)
 - if the stronger thread is RT, immediate preemption
 - if the stronger thread is not RT, at next convenient time
- A thread becomes non-computable, i.e.
 - it goes to sleep
 - it is stopped
 - it exits(Voluntary)

Student Notes

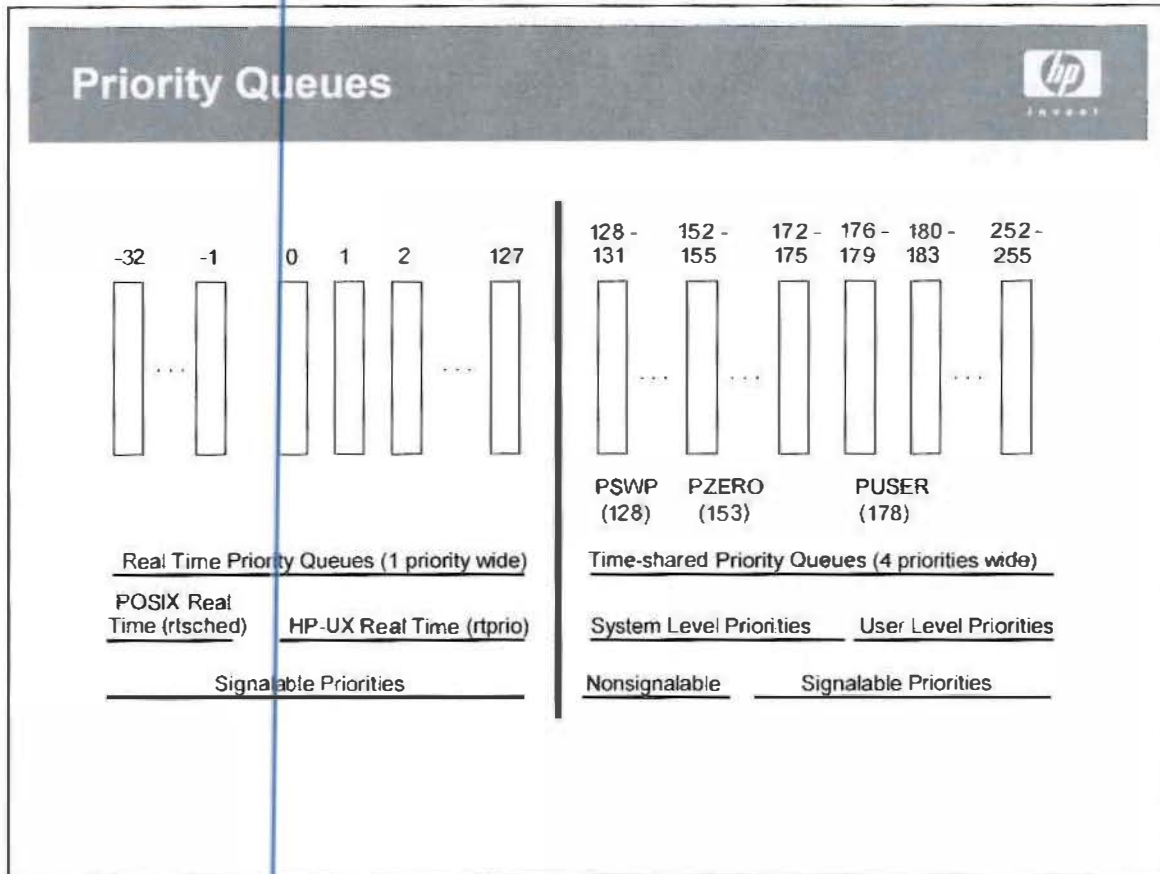
A **context switch** is the mechanism by which the kernel stops the execution of one process and begins execution of another. A context switch occurs under the circumstances shown on the slide.

There are two types of context switches: **forced** and **voluntary**. A forced context switch occurs when the process is forced to give up the CPU before it is ready. These include timeslice expiration or a stronger priority process becoming runnable.

A voluntary context switch occurs when the process itself gives up the CPU without using its full timeslice. This happens when the process exits, or puts itself to sleep (waiting on a resource), or puts itself into a stopped state (debugging).

The **glance** tool distinguishes between forced and voluntary context switches on a per process basis.

4-9. SLIDE: Priority Queues



Student Notes

Every process has a priority associated with it at creation time. These priorities determine the order in which processes execute on the CPU. Processes with the weakest priority number always execute before processes with stronger numbers. In UNIX, stronger priorities are represented by smaller numbers and weaker priorities are represented by larger numbers.

HP-UX uses adjustable priorities to schedule its time slicing for general timeshare processes generated by all users (priorities 128-255). By that we mean, a process's priority can be adjusted, up or down, by the kernel, according to how "favored" a process might be. In general, the more a process executes, the less favorable it will be treated by the kernel.

However, since HP-UX also supports real-time processing, it must include priority-based scheduling for those processes (priorities 0-127). As of HP-UX 10.X, support is also provided for POSIX real-time processes (priorities -32 through -1). The `/usr/include/sys/param.h` file contains some extra information on the priorities used in the system.

Each processor in an HP system has its own run queue. Each run queue is further broken down into multiple priority queues, to make it easier for that processor to select the most deserving process to run.

Real-Time Process Priorities

Real-time priority queues are one wide, i.e. each queue represents one priority value. The strongest priority real-time process preempts all others (of weaker priority) and runs until it sleeps or exits or is preempted by a stronger or timesliced by an equal real-time process. Equal priority real-time processes run in a round robin fashion. A process can be made to run with a real-time priority by using the `rtprio(1)` or `rtsched(1)` command. The `rtsched` command can also be used to disable timeslicing for a particular process, by assigning it a different scheduling policy.

Because a real-time process will execute at the expense of all time-share processes, make sure that you consider the impact on your users before invoking the command. A CPU-bound, real-time process will halt all other use of the system.

A POSIX real-time process (`rtpr`) runs on HP-UX at priority -32.

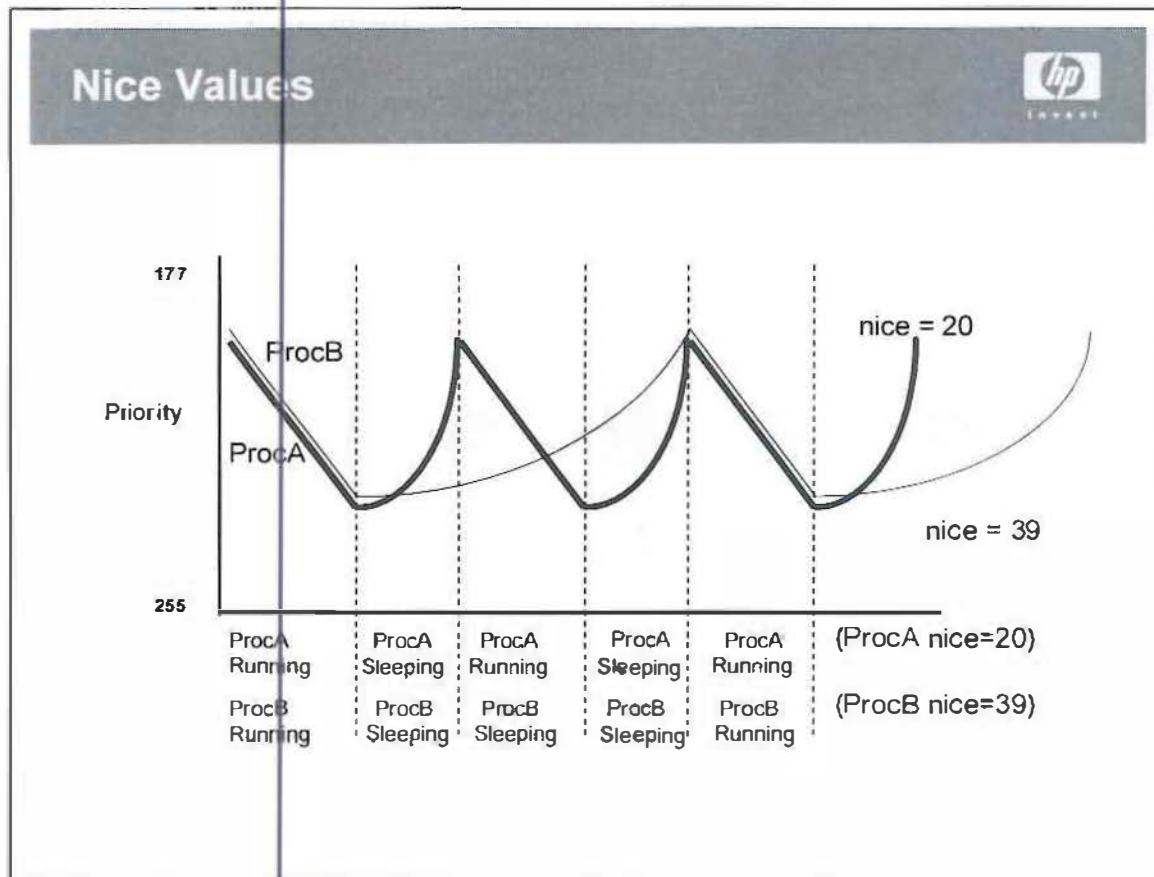
Time Share Process Priorities

Timeshare priority queues are four-wide, i.e. each priority queue represents four, adjacent priority values. For example, the first timeshare priority queue is used by processes with priorities of 128, 129, 130, and 131.

Timeshare processes are grouped into system and user processes. Priorities 128-177 are reserved for runnable system processes and sleeping processes (both system and user), and priorities 178-255 are for runnable user processes.

A nice value is assigned to a timeshare process, which will be used in the calculation of a new priority for the process. This value will be used by the kernel to help determine how to "adjust" the priority of the process. Nice values have no effect on real-time processes.

4-10. SLIDE: Nice Values



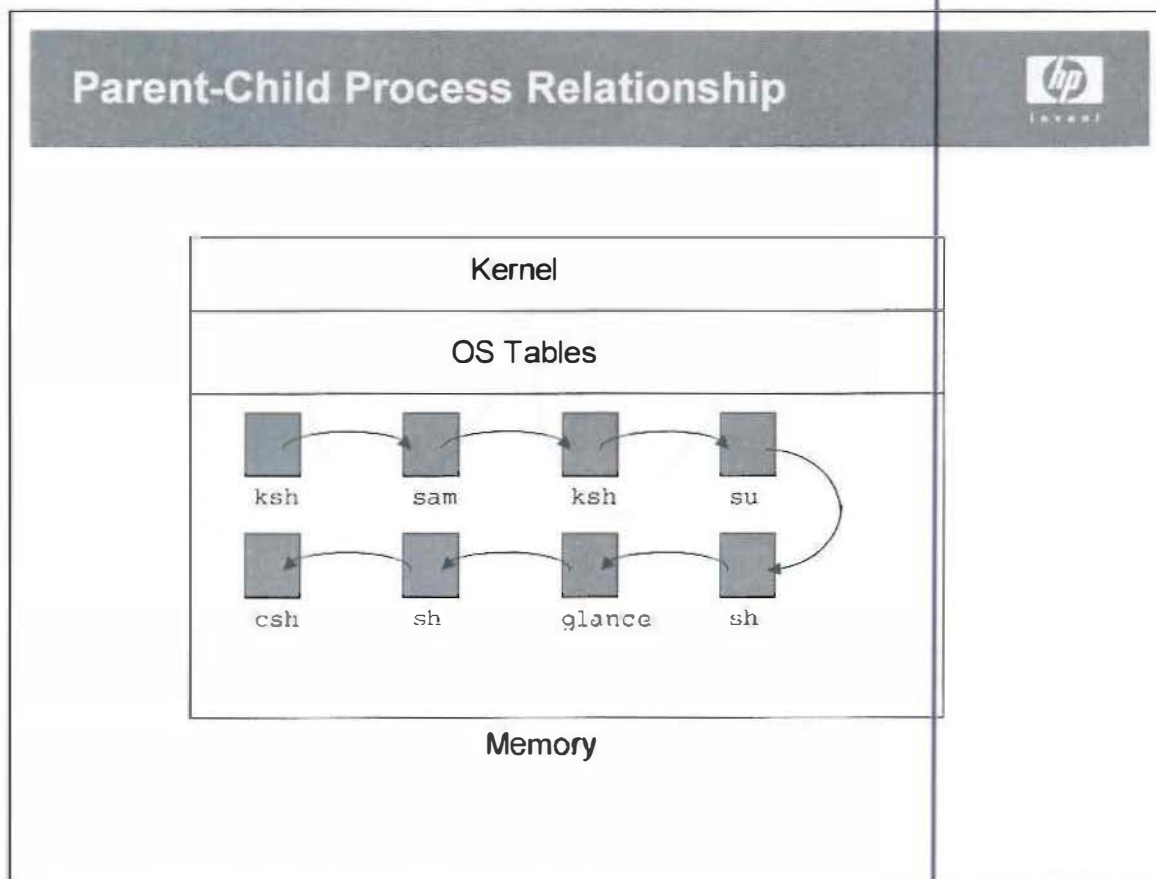
Student Notes

Time shared processes are all initially assigned the priority of the parent when they are spawned. The user can make modifications to how much the kernel "favors" a process with the nice value. Timeshare processes lose priority as they execute, and regain priority as they wait their turns. *The rate at which a process loses priority is linear, but the rate at which it regains priority is exponential.*

A process's nice value is used as a factor in calculating how fast a process regains priority. The nice value is the only control a user has to give greater or less favor to a time share process. The default nice value is 20. Therefore, to make a process run at a weaker priority, it should be assigned a higher nice value (maximum value 39). The *superuser* can assign a lower nice value to a process (minimum value 0), effectively giving it a stronger priority.

Si j'ai un processeur de PPI 30
je m'occupe de lui
PID

4-11. SLIDE: Parent-Child Process Relationship



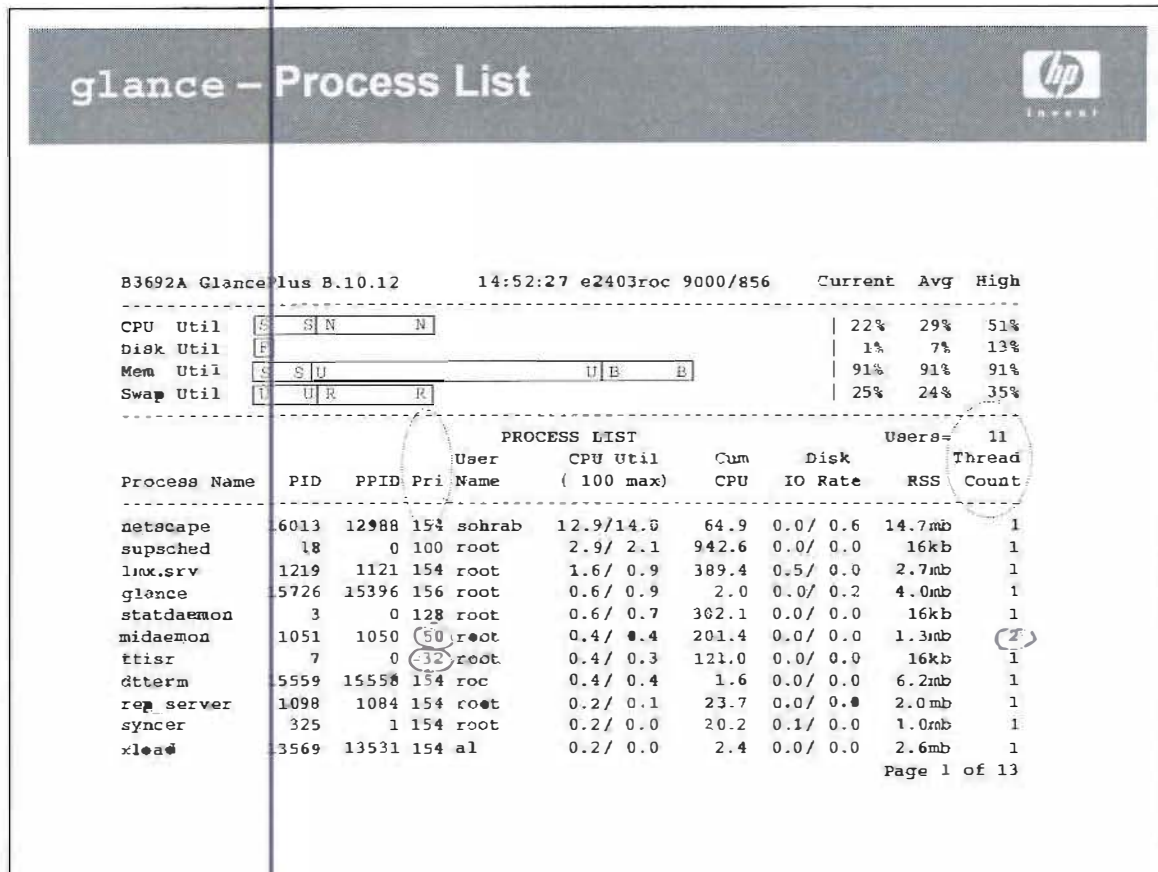
Student Notes

One item to keep in mind related to process management is the relationship between parent and child processes. Every process started from a terminal window on the system has a parent process that spawns it. The parent process does not terminate once a child is spawned. Instead, it goes to sleep waiting for the child to terminate from its execution.

If a child process does not exit properly, for example, if it spawns a new process rather than exiting to its parent, then the system could end up with many processes sleeping in memory and using proc table entries unnecessarily.

The example in the slide shows a `ksh` shell that spawns a `sam` process. Within `sam`, the system administrator shells out to `su` to a regular user. Once in the login shell, the user starts `glance`. From within `glance`, they shell out, and now decide they'd rather be in a `csh` shell. This string of events caused eight different processes to be started. If the user decides he wants to return to `sam` by typing `sam`, would the previous `sam` process be reactivated, or would a new `sam` process be spawned? (Answer: A new `sam` process is spawned).

4-12. SLIDE: glance — Process List



Student Notes

The next four slides are designed to illustrate how the management of processes can be monitored through glance. Topics just covered (like kernel versus user CPU time, process components, process wait states, nice values, and process priorities) can all be viewed through glance.

The first Global Bar graph, which displays on every glance screen, is the CPU Util. This displays how the CPU is being distributed.

- S = System or Kernel Time
- N = User Time (executing processes who have had their nice value set greater than 20. (21-39))
- U = User Time (executing processes with a nice value of 20)
- A = User Time (executing processes who have had their nice value set less than 20 (0 - 19). In other words: Anti-nice.
- R = Real Time (executing processes with priorities 127 and less)

Module 4

Process Management

The Process List screen (**g** key), as shown on the slide, can be used to see *process priorities*. The order in which the processes are displayed can be configured (**o** key) to display by CPU usage, memory usage, or disk I/O activity.

In HP-UX version 10.X, the *thread count* column was the *blocked on* column. The *blocked on* information can still be obtained by looking at the individual processes' resource summary screens.

4-13. SLIDE: glance — Individual Process

glance — Individual Process



B3692A GlancePlus B.10.12
15:17:52 e2403roc 9000/856
Current Avg High

CPU Util	S S N N	22%	29%	51%
Disk Util	F	1%	7%	13%
Mem Util	S S U U B B	91%	91%	91%
Swap Util	U U R R	25%	24%	35%

Resource Usage for PID: 16013, netscape PPID: 12988 euid: 520 User:sohrab

CPU Usage (sec) :	3.38	Log Reads :	166	Wait Reason :	SLEEP
User/Nice/RT CPU:	2.43	Log Writes:	75	Total RSS/VSS:	22.4mb/ 28.3mb
System CPU :	0.73	Phy Reads :	4	Traps / Vfaults:	414/ 8
Interrupt CPU :	0.14	Phy Writes:	61	Faults Mem/Disk:	0/ 0
Cont Switch CPU :	0.08	FS Reads :	4	Deactivations :	0
Scheduler :	HPUX	FS Writes :	29	Forks & Vforks :	0
Priority :	154	VM Reads :	0	Signals Recd :	339
Nice Value :	24	VM Writes :	0	Messg Sent/Recd :	775/ 1358
Dispatches :	1307	Sys Reads :	0	Other Log Rd/Wt:	3924/ 957
Forced CSwitch :	460	Sys Writes:	32	Other Phy Rd/Wt:	0/ 0
VoluntaryCSwitch:	814	Raw Reads :	0	Proc Start Time	
Running CPU :	0	Raw Writes:	0	Fri Feb 6 15:14:45 1998	
CPU Switches :	0	Bytes Xfer:	410kb		

Student Notes

From the Process List screen, an individual process can be selected for further analysis (s key). The above slide shows some of the additional details available when analyzing a process further.

Items of interest from the Individual Process screen include the process's nice value, the number of Forced versus Voluntary context switches, the current Wait reason, and the Parent PID.

Sur la cpu un processus est exécuté plus libre par le système ... ce qu'on appelle

4-14. SLIDE: glance — Process Memory Regions

glance – Process Memory Regions


```

B3692A GlancePlus B.10.12      10:17:41 e2403roc 9000/856      Current Avg High
-----
CPU Util   S S N N                | 22%  29%  51%
Disk Util  F                      |  1%   7%  13%
Mem Util   S S U U U H B          | 91%  91%  91%
Swap Util  U U R R                | 25%  24%  35%
-----
Memory Regions for PID: 16013, netscape      PPID: 14061  euid: 520  User:sohrab

Type      RefCt   RSS    VSS    Locked  File Name
-----
NULLDR/Shared  64    4kb    4kb     0kb    <nullhref>
TEXT /Shared    3   4.3mb   9.5mb    0kb    /opt/_/netscape-bin
DATA /Priv      1   5.8mb   8.6mb    0kb    /opt/_/netscape-bin
MEMMAP/Priv     1    4kb    20kb    0kb    /opt/_/netscape-bin
MEMMAP/Priv     1   36kb   36kb    0kb    /opt/_/netscape-bin
MEMMAP/Priv     1   12kb   12kb    0kb    <memmap>
STACK /Priv     1   28kb   28kb    0kb    <stack>
UAREA /Priv     1   16kb   16kb    0kb    <uarea>
LIBTXT/Shared   85   56kb   60kb    0kb    /usr/lib/dld/sl

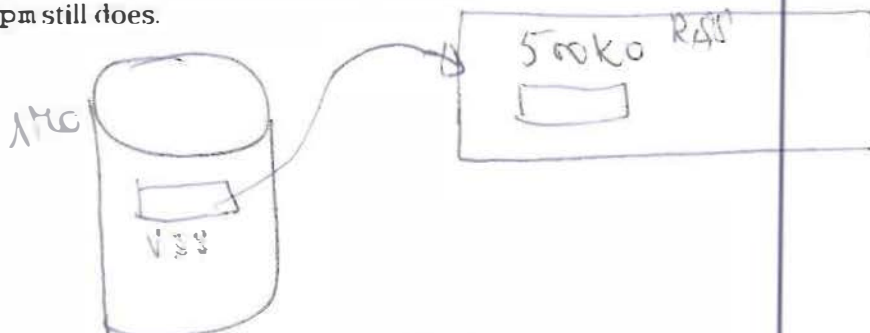
Text RSS/VSS: 4.3mb/9.5mb  Data RSS/VSS: 5.8mb/8.6mb  Stack RSS/VSS: 28kb/ 28kb
Shmem RSS/VSS: 0kb/ 0kb   Other RSS/VSS: 4.1mb/5.7mb
  
```

Student Notes

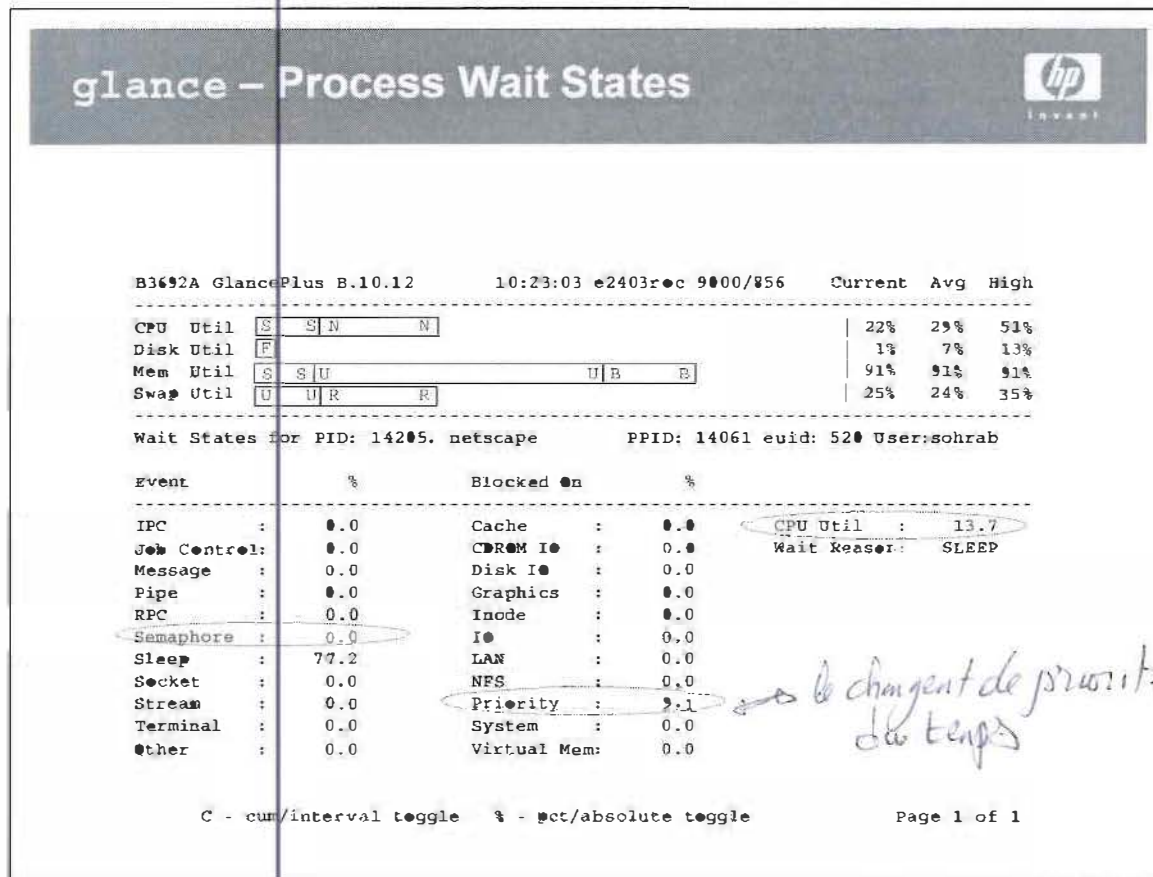
From the Individual Process screen, the memory regions (i.e. process components) corresponding to that process can be viewed (**M** key). The above slide shows the memory regions for the currently selected process.

Items of interest from the Memory Region screen include the location of the process's Text, Data, Stack, and U-Area, along with its Shared/Private flag, its Resident Set Size and Virtual Set Size, and its reference count. If the process is associated with Memory Map files (MEMMAP), Shared Libraries (LIBTXT), or Shared Memory Segments (SHMEM), these will be displayed.

In HP-UX version 11.X, **glance** no longer displays the addresses of each memory region. However, **gpm** still does.



4-15. SLIDE: glance — Process Wait States



Student Notes

From the Process List screen, the process wait states can be viewed (W key). The above slide shows the categories of wait states and where/what the selected process has waited on.

Items of interest from the Process Wait State screen include the percentage of time the process has spent in each of the possible wait state categories.

4-16. LAB: Process Management

Directions

The following lab is designed to manage a group of processes. This includes observing the parent-child relationship and modifying process nice values (and thus indirectly priorities) with the `nice/renice` command.

Modifying Process Priorities

This portion of the lab uses `glance` to monitor and modify nice values of competing processes.

1. Change directory to `/home/h4262/baseline`.

```
# cd /home/h4262/baseline
```

2. Start seven **long** processes in the background.

```
# ./long & ./long & ./long & ./long & ./long & ./long & ./long &
```

3. Start a `glance` session. Answer the following questions.

How much CPU time is each **long** process receiving? _____ sec, _____ %

How are the processes being context switched (forced or voluntary)? _____

How many times over the interval is the process being dispatched? _____

What is the ratio of system CPU time to user CPU time? _____

What are the processes being blocked on? _____

What are the nice values for the processes? _____

4. Select one of the processes and favor it by giving it a more favorable nice value.

What is the PID of the process being favored? _____

To change the processes nice value, enter:

```
# renice -n -5 <PID of selected process>
```

Watch that process's percentage of the CPU over several display intervals with `glance` or `top`. What effect did it have on the process? _____

5. Select another long process and set the nice value to 30.

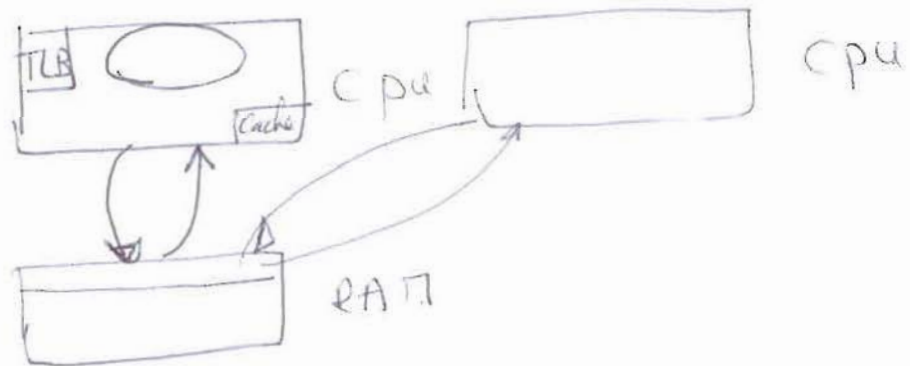
```
# renice -n 10 <PID of another selected process>
```

What effect did that have on that process? _____

6. You can either let the processes finish up on their own as the next module is covered, or you can kill them now with:

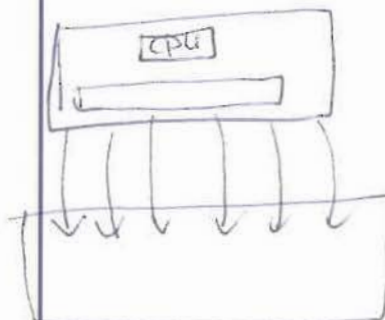
```
# kill $(ps -el | grep long | cut -c18-22)
```

RISC = power pc / ultra sparc / PA RISC
Reduce instruction cycle



Method IPIC

Instruction parallel cycle state stream



Attention: mais il faut que l'application soit multithread

Hp a investi plus que DEC car ces derniers ont investi sur la recherche
Hp a mis un firmware permettant d'utiliser le cache et le co-processeur au niveau CPU

<http://education.hp.com>

4-27

H4262S C.00

© 2004 Hewlett-Packard Development Company, L.P.

Cache CPU: 32 Mo de cache CPU pour le serveur

Tpm : transaction per minute (normalisation Tpm)

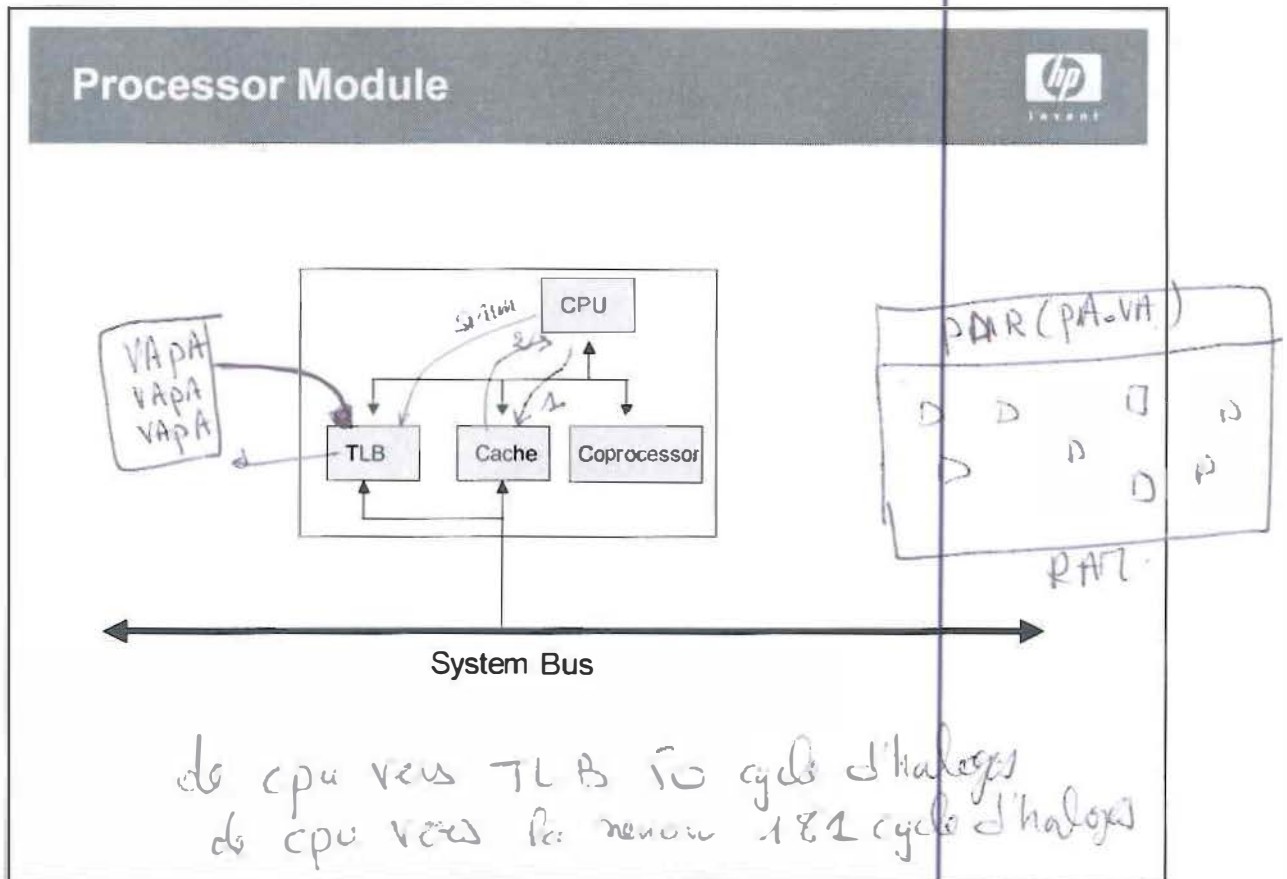
Module 5 — CPU Management

Objectives

Upon completion of this module, you will be able to do the following:

- Describe the components of the processor module.
- Describe how the TLB and CPU cache are used.
- List four CPU related metrics.
- Identify how to monitor CPU activity.
- Discuss how best to use the performance tools to diagnose CPU problems.
- Specify appropriate corrections for CPU bottlenecks.

5-1. SLIDE: Processor Module



Student Notes

A typical HP processor module consists of a central processing unit (CPU), a cache, a translation lookaside buffer (TLB), and a coprocessor. These components are connected via internal processor busses, with the entire processor module being connected to the system bus.

The **cache** is made up of very high-speed memory chips. Cache can be accessed in one CPU cycle. Its contents are instructions and data that recently have been or are anticipated to be used soon by the CPU. Cache size varies between processors. The size of the cache can have a big effect on system performance.

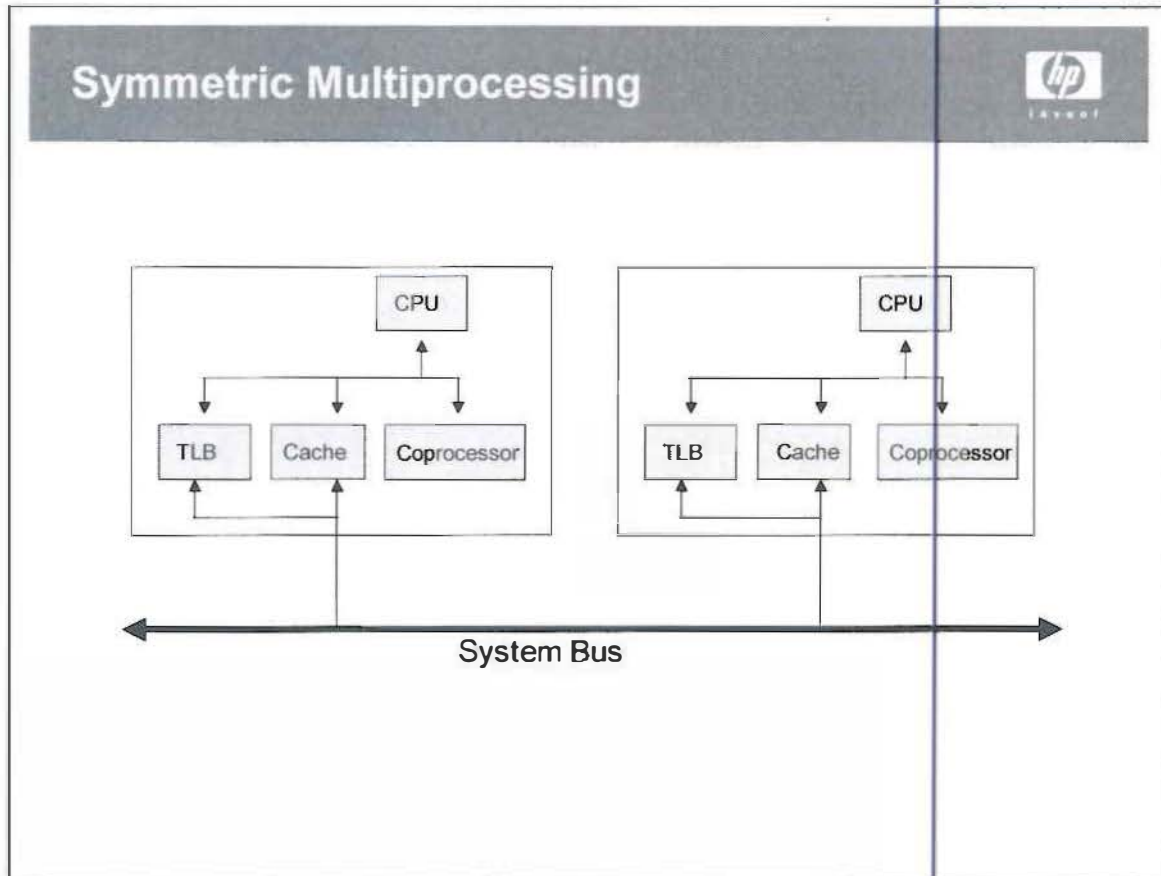
The **translation lookaside buffer (TLB)** is used to translate virtual addresses into physical addresses. It is a high-speed cache whose entries consist of pairs of recently accessed virtual addresses and their associated physical addresses, along with access rights and an access ID. The TLB is a subset of a system-wide translation table (page directory) that is held in memory. TLB size also affects system performance, and different HP 9000 processors have different TLB sizes.

The address translations kept in the TLB enable us to locate the appropriate data and instructions in the memory. The memory is accessed via the physical address. Without the translation in the TLB, we would not be able to find the information in the memory.

Note these other points regarding the TLB:

- Each process has a unique virtual address space.
- Each TLB entry refers to a *page* of memory, *not* a single location. In all 64-bit architectures used by HP, pages are fundamentally 4KB in size, but can be any multiple of 4K under various circumstances – to reduce the number of entries needed in the TLB.

5-2. SLIDE: Symmetric Multiprocessing

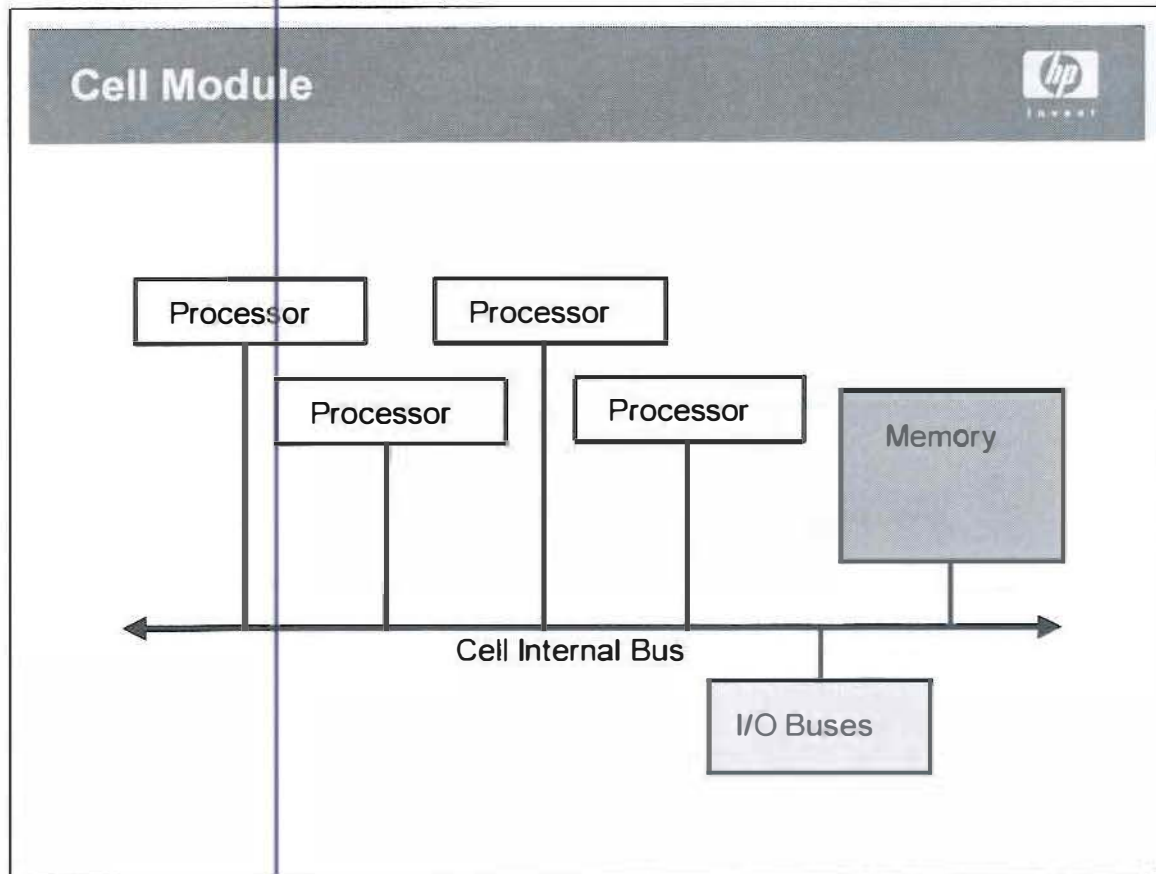


Student Notes

Symmetric Multiprocessing (SMP) refers to systems containing two or more processor units. SMP is implemented on all Hewlett-Packard workstations and servers capable of supporting more than one CPU.

Each processor on an SMP system has exactly the same characteristics, including the same processing unit, the same CPU cache design, and the same size translation lookaside buffer (TLB).

5-3. SLIDE: Cell Module



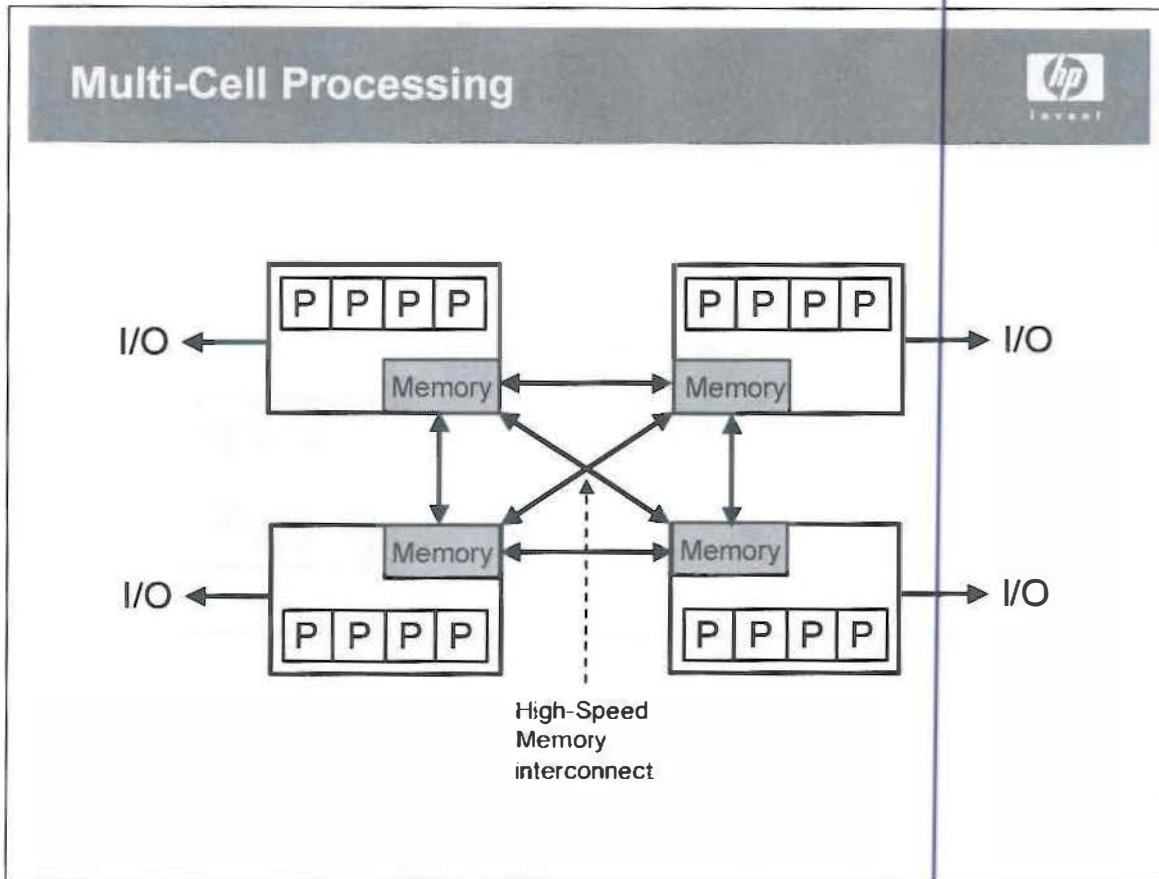
Student Notes

A more recent design of HP systems is based on the "cell" architecture. In a cell, there are multiple processors, some memory and some I/O buses. Each cell could act as an independent SMP system, or as part of a collection of cells, forming a larger SMP system.

Each processor in the cell has the same access speed (or latency) to the memory within the same cell. However, if one of those processors would have to access a location in the memory of a different cell, the latency would be greater.

Each processor within the cell does have its own cache memory and TLB. Each processor has equal access to the I/O buses that are part of the same cell. They may also have access (with somewhat greater delays) to the I/O of other cells in the same system.

5-4. SLIDE: Multi-Cell Processing



Student Notes

The best example HP currently has of a SMP using cell architecture is the Superdome. Here we find 4 cells, each with four processors, some memory, and some I/O buses. Each cell could be configured (using Node Partitioning or NPars) into a separate and individual system – capable of booting its own operating system. It would be functionally apart from the other cells. The only way that the operating system on that cell could communicate with the software running on any other cell would be through a network interface.

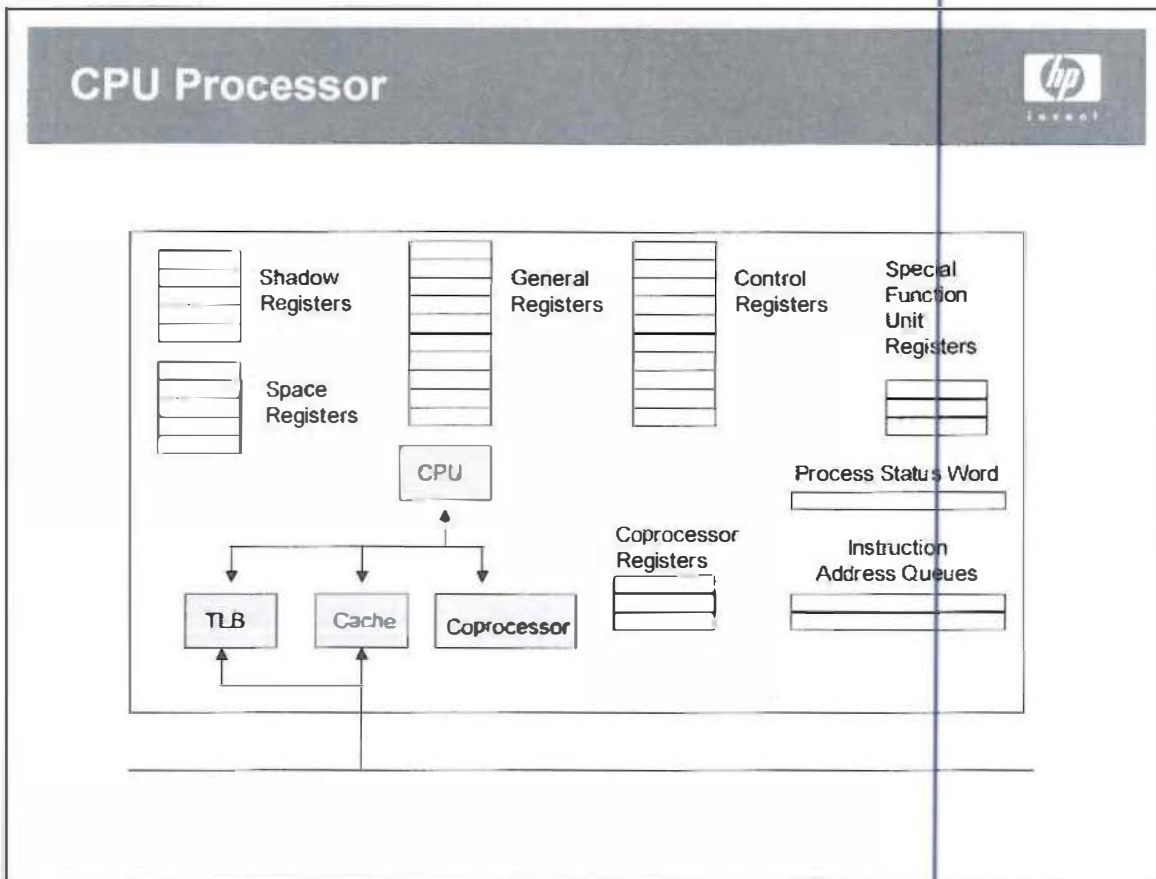
On the other hand, multiple cells could be configured to act as a unit. They would pool their resources and boot a single operating system. They would seamlessly act as a SMP system.

This architecture gives the customer and the system administrator tremendous flexibility in how to set up their hardware. They could even change it relatively easily from one configuration to another as their needs changed.

On a wider range of systems, you may be using Virtual Partitioning (VPar). There are similar to NPars, but are not limited to cell boundaries and are handled entirely by software. A system could use both NPars and VPar at the same time. Using software, processors can be moved from one VPar to another.

Finally, on an even wider range of systems, we have the concept of processor sets (psets). Multiple psets could exist within the same partition (either NPar or VPar). Each pset would be set aside for use by a particular application or group of applications. Using software, psets can be created and removed, and processors could be moved from one pset to another.

5-5. SLIDE: CPU Processor



Student Notes

The CPU ultimately is responsible for your system speed. The kernel loads the process text for the CPU to execute. The processor module has many Registers, which assist in the execution of instructions. The definition of all these registers is beyond the scope of this course. The primary objective of this module is to focus on CPU clock speed, the size of the CPU cache, and the effects of the TLB related to overall system performance.

Each HP 9000 server and workstation has a chip at its heart. The latest version PA-RISC chip is the 64-bit, PA-8xxx. HP has also introduced systems using the 64-bit, IA-64 Itanium chip. A selection of the range of current systems is listed on the following pages. Note the difference not only in clock speeds, but also in cache size.

The following tables list the specifics of several HP-UX servers and workstations. It is very difficult to keep a list of this nature up to date in training materials but it has been included merely to demonstrate the wide variety of system characteristics present in the HP computing products family.

Business Servers

Model	No. of CPUs	Clock Speed	Max. RAM (GB)	Cache (KB)	I/O Slots
rp3410-2 (PA-8800)	2	800MHz	6	1.5MB(L1) 32MB(L2)	2-PCI (64-bit)
rp3440-4 (PA-8800)	4	1 GHz	24	1.5MB(L1) 32MB(L2)	4 PCI (64-bit)
rp4440-8 (PA-8800)	8	1 GHz	64	1.5MB(L1) 32MB(L2)	6 PCI
rp7420-16 (PA-8800)	16 (2 cells)	1 GHz	64	1.5MB(L1) 32MB(L2)	15 PCI
rp8420-32 (PA-8800)	32 (4 cells)	1 GHz	128	1.5MB(L1) 32MB(L2)	16 PCI
Superdome (PA-8800)	128 (16 cells)	1 GHz	1024	1.5MB(L1) 32MB(L2)	192 PCI
rx1600 (Itanium 2)	2	1 GHz	16	1.5MB(L3)	0/1/1 PCI *
rx2600 (Itanium 2)	2	1.5 GHz	24	6MB(L3)	0/4/0 PCI *
rx4640 (Itanium 2)	4	1.5 GHz	64	6MB(L3)	0/4/2 PCI *
rx5670 (Itanium 2)	4	1.5 GHz	96	6MB(L3)	0/6/3 PCI *
rx7620 (Itanium 2)	8 (2 cells)	1.5 GHz	64	6MB(L3)	15 PCI (128-bit)
rx8620 (Itanium 2)	16 (4 cells)	1.5 GHz	128	6MB(L3)	16+16 PCI (128-bit)
Superdome (Itanium 2)	64 (16 cells)	1.5 GHz	512	6MB(L3)	0/128/64 PCI *

Module 5
CPU Management

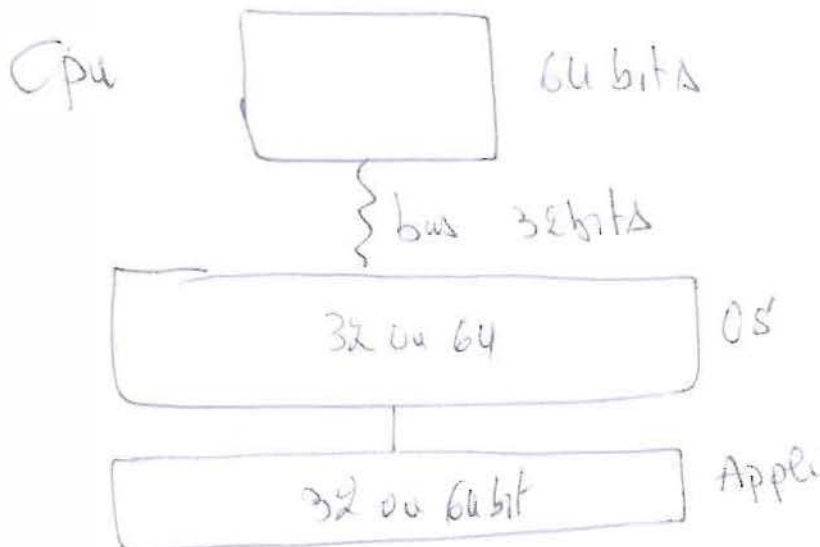
Workstations

Model	No. of CPUs	Clock Speed	Max. RAM (GB)	Cache (KB)	I/O Slots
B2600 (PA-8600)	1	500 MHz	4	512/1024	2/2/0 PCI *
B3700 (PA-8700)	1	750 MHz	8	768/1536	2/3/1 PCI *
C3750 (PA-8700+)	1	875 MHz	8	768/1536	2/3/1 PCI *
J6750 (PA-8700+)	2	875 MHz	16	768/1536	0/0/3 PCI *
zx2000 (Itanium 2)	1	1.1 GHz	8	1536(L3)	5 PCI - 1 AGP
zx6000 (Itanium 2)	2	1.5 GHz	24	6144(L3)	3 PCI - 1 AGP

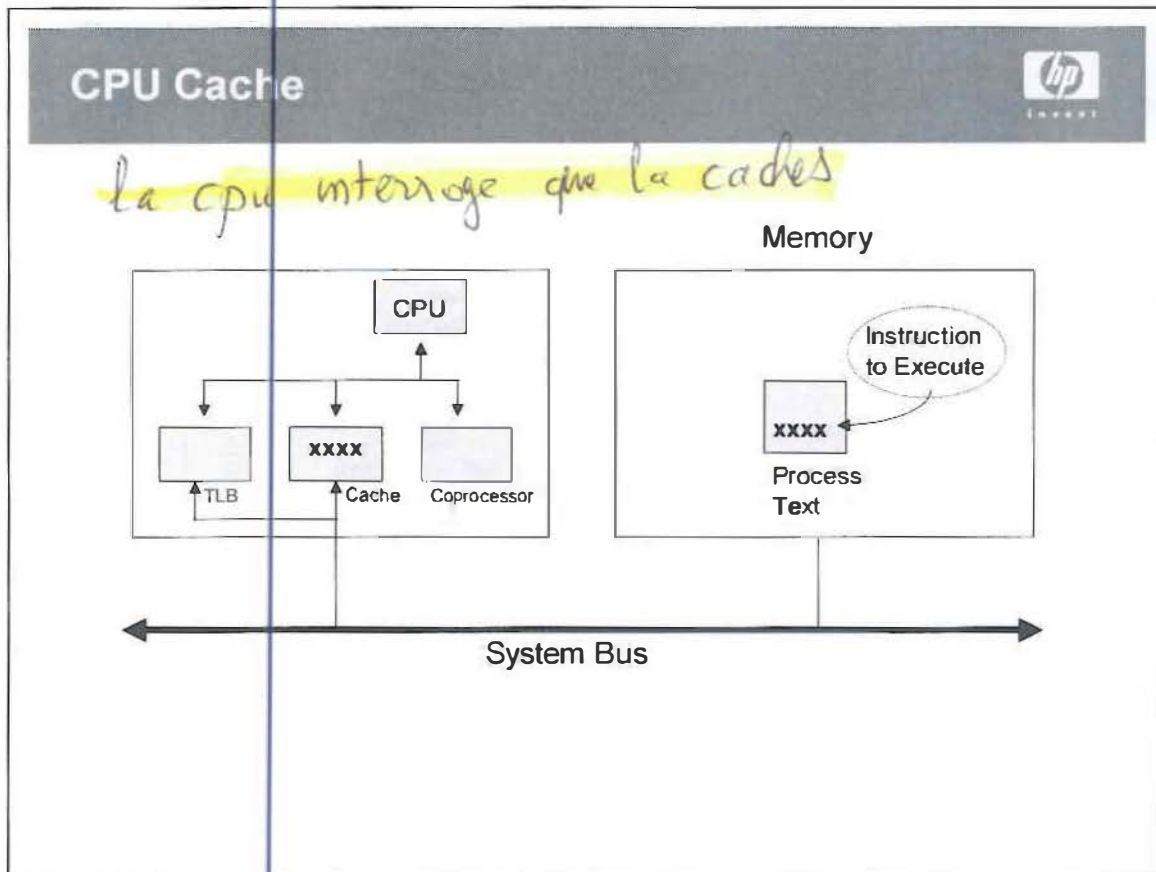
* 2/3/1 means 2 32-bit PCI's, 3 64-bit PCI's and 1 128-bit PCI.

All Itanium 2 processors include 32KB of L1 cache and 256KB of L2 cache.

To determine the specifics of your system, refer on-line to <http://www.hp.com/go/enterprise>, select "Products Index" and scroll down to select your system platform name [i.e. J-Class (HP 9000)]. This will display the "Product Information" screen for the selected hardware.



5-6. SLIDE: CPU Cache



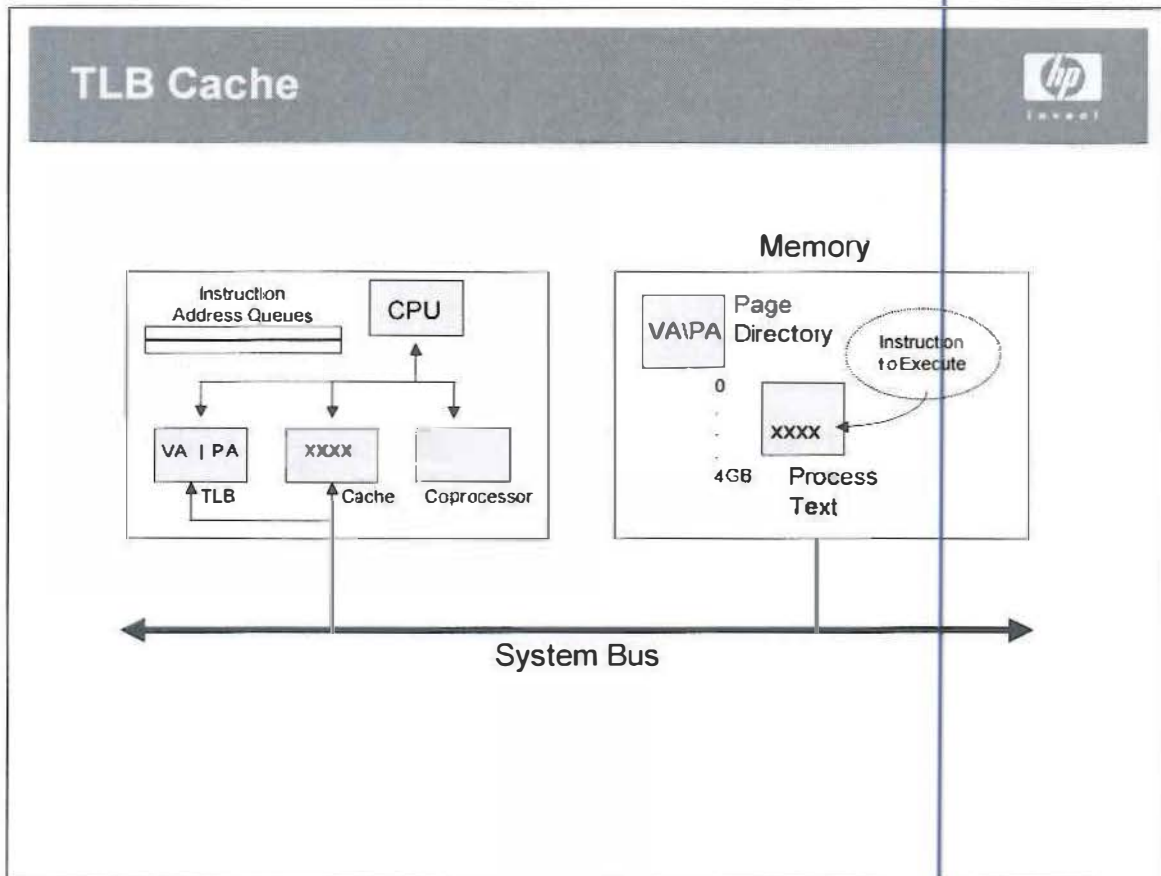
Student Notes

The CPU loads instructions from memory and runs multiple instructions per cycle. To minimize the time that the CPU spends waiting for instructions and data, the CPU uses a cache.

The cache is a very high-speed memory that can be accessed in one CPU cycle with the contents being a subset of the contents of main memory. As the CPU requires instructions and data, they are loaded into the cache. The size of the cache has a large bearing on how busy the CPU is kept. The larger the cache, the more likely it is that it will contain the instructions and data to be executed.

Most current processors support multi-level caches. The Level 1 cache (L1) is the fastest – operating at the same speed as the CPU. It is relatively small. The Level 2 cache (L2) operates at one-half the speed of the CPU. It is somewhat larger. The IA-64 has a Level 3 cache (L3) that is even larger and slower.

5-7. SLIDE: TLB Cache



Student Notes

All 32-bit programs view their address space as starting at address 0, and ending at address 4 GB. All addresses referenced by the program are referenced relative to this address space. This is referred to as the program's **virtual address space**.

A program's **physical address** is the address location in physical memory where the program is loaded at execution time.

When the CPU executes a program, it is presented with the virtual address containing the instruction to be executed. In order to fetch this instruction from physical memory, the CPU must convert the virtual address (VA) into the corresponding physical address (PA). To do this, the CPU checks the TLB. If the VA->PA is present, it then knows the PA in memory of the instruction. If the VA is not present, it then needs to fetch the information from the PDIR (Page DIRectory) table in memory. This memory fetch of the PDIR table is relatively expensive from a performance standpoint.

Once the PA is known, the CPU then checks the Instruction Cache on the CPU for the PA. If the PA is present, it then loads the instruction straight from Instruction Cache. If not present, it then needs to fetch the instruction from memory, which is relatively expensive (performance-wise).

The size of the TLB is anywhere from 96 to 160 entries (each entry points to a variable-sized memory page) on a PA-RISC and an IA-64.

5-8. SLIDE: TLB, Cache, and Memory

TLB, Cache, and Memory			
TLB	Cache	Memory	Consequence
Hit	Hit	Hit	1 CPU cycle fetch ✓
Hit	Miss	Hit	Data/instruction memory fetch 50 cycles sent to TLB
Miss	X	Hit	PDIR memory fetch 131 cycle & maybe
Miss	X	Miss	Page fault → need swap & verify it check the info

X = Don't Care

Student Notes

The slide shows some of the permutations of hits and misses on memory, cache, and the TLB, as well as the consequences of each. The best situation is when the VA has an entry in the TLB, and the corresponding PA has an entry in the CPU cache. This allows the instruction or data to be present to the CPU in one clock cycle.

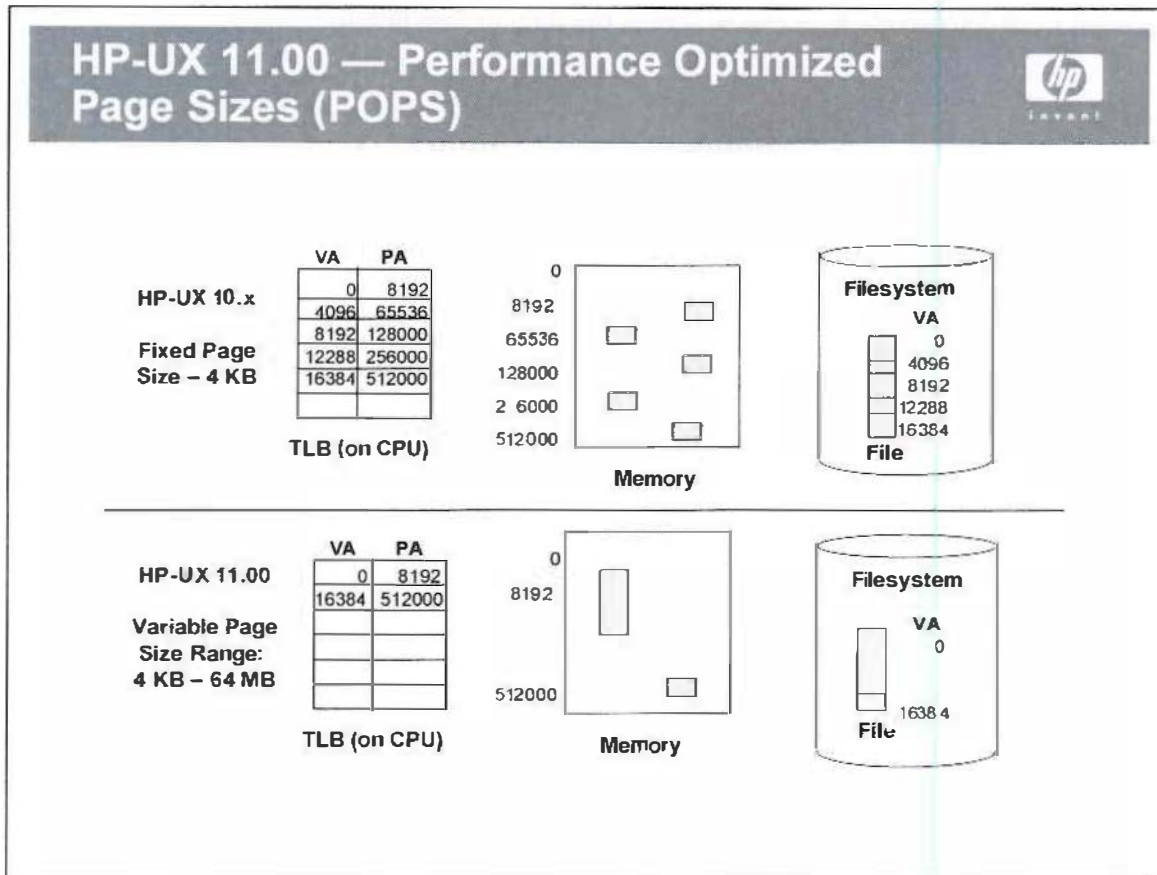
The next-best scenario, would be to have a hit on the TLB, but a miss on the CPU cache. An example number of clock cycles to fetch a PA from memory to the CPU cache is 50 clock cycles.

Another scenario would be to have a miss on the TLB, but a hit on the CPU cache. The miss on the TLB requires the PDIR table in memory to be searched, and an appropriate entry to be loaded into the TLB. This takes a variable number of cycles to perform. On one model the average was 131 clock cycles. Therefore, a miss on the TLB is more expensive than a miss on CPU cache.

A miss on both the TLB and the CPU cache would translate into 131 + 50 or 181 clock cycles on average to access the instruction or data that the CPU needs. This could have been accessed in 1 clock cycle had the VA been in TLB, and the PA been in CPU cache.

The worst scenario, performance-wise, is not having the instruction or data loaded in memory at all. In this case, a page fault would occur to retrieve the information from disk. Assuming a 1-GHz clock, a 10-ns disk transfer rate, and an idle disk drive, this would correspond to 10,000,000 clock cycles to access the data or instruction.

5-9. SLIDE: HP-UX — Performance Optimized Page Sizes



Student Notes

HP-UX 11.00 is the first release of the operating system to have general support for **performance optimized page sizes (POPS)**, also known as variable page sizes. Partial support for variable memory page sizes has existed since HP-UX 10.20.

HP-UX 11.00 allows customers to configure executables to use specific performance optimized page sizes, based on the program's text and data sizes. Page sizes can be selected from a range of 4 KB to 4 GB. The use of performance optimized page sizing can significantly increase performance of applications that have very large data or instruction sets.

NOTE: Performance-optimized page sizing works on PA-8000-based and IA-64-based systems.

Fixed Page Sizes (Prior to 11.00)

Prior to HP-UX 11.00, all page sizes were fixed at 4 KB. As a program executed, each 4 KB page would be mapped into physical memory, and a TLB entry would be created to map the virtual address corresponding to that page to the physical memory address. Selected models

had a few "Block" TLB entries, which could map multiple pages into a single entry, if the pages were contiguous in both virtual and physical address spaces. These entries were reserved for mapping the kernel, the I/O pages, and other segments that were locked into memory.

At some point, the TLB would become full, and the virtual-to-physical address mapping would only be stored in the PDIR table in memory, not in the TLB on the CPU. This meant that if a virtual address needed to be translated, there would be a chance that the address would *not* have an entry in the TLB, and time would have to be spent to look up the address within the PDIR table in memory. This handling of the TLB miss was expensive in terms of performance.

Performance Optimized Page Sizes (11.00 and Beyond)

With the release of HP-UX 11.00, support for variable page sizes is available. With POPS, a larger portion of the process's virtual address space can be referenced within a single page or within a few, large pages. Therefore, a larger portion of the process can be referenced with much fewer TLB entries. Below are two tables showing what sizes of pages are available in the PA-RISC and the IA-64 architectures.

PA-RISC

4K
-
16K
64K
256K
1M
4M
16M
64M
256M
1G
-

IA-64

4K
8K
16K
64K
256K
1M
4M
16M
64M
256M
-
4G

max de page [la partie max de data dupron

Affecting Page Sizes

There are two methods of affecting page size in a process. One is through tunable kernel parameters. `vps_pagesize` determines what the "default" page size will be with no other information. The size is given in 1K units and the setting is typically 4. `vps_ceiling` determines how large the kernel can "promote" a page size for a process, if it notices that a process is getting a very large number of TLB misses. The default setting for this is 16 (1K).

The second method is done by the system administrator. A command, `chattr`, can be used to provide the kernel with a hint of what page sizes would work best with this process. Following is an example of this command.

```
chattr -pi 16 -pd 256 /opt/app/bin/app
```

The above command would hint to the kernel that this process would best execute with 16K pages for the instructions (text) and 256K pages for the data. This hint would be stored in the

Module 5 CPU Management

header of the executable file and be visible to the kernel whenever the program was invoked. The kernel would do its best to see that the hint is followed. However, if memory pressure exists, the kernel may not be able to honor the request and may end up "demoting" the size of the page to be able to manage it in memory. There is a third tunable parameter, **vps_chattr_ceiling**, that determines the maximum value a **chattr** command can assign to an executable file.

vps-ceiling (kernel)
vps-chattr-ceiling

Ksh : 12 Mo → cve (ulimit) vérifier cette valeur si il
est ne dépasse pas 12 Mo

5-10. SLIDE: CPU — Metrics to Monitor Systemwide

CPU — Metrics to Monitor Systemwide



- User CPU utilization
- Nice/Anti-Nice utilization
- Real time processes
- System CPU utilization
- System call rate
- Context switch rate
- Idle CPU utilization
- CPU run queues (load averages)

Student Notes

The load on the CPU can be monitored in a number of different ways. There are multiple tools and multiple metrics that monitor CPU performance.

User CPU Utilization

This is the percentage of time the CPU spent running in user mode. This corresponds to executing code within user processes, as opposed to code within the kernel. It is better to see user CPU utilization higher than system CPU utilization (preferably two to three times higher).

Nice/Anti-Nice Utilization

This is the percentage of time the CPU spent running user processes with nice values of 21-39 (Nice) or 0-19 (Anti-Nice). This is typically included in USER CPU utilization, but some tools, like **glance**, track this separately to see how much CPU time is being spent on weaker or stronger priority processes.

Real Time Processes

This is the amount of time spent executing real time processes that are running on the system. Real time processes get the CPU immediately when they are ready to execute, and can have a big impact on the performance of time-shared processes.

System CPU Utilization

This is the percentage of time the CPU spent running in system (or kernel) mode. This corresponds to executing code within the kernel. We have to have some kernel time just to do minimum management on the system. However, excessive time spent managing the system is bad for performance. Excessive system CPU utilization is considered to be when system utilization is greater than the user utilization.

System Call Rate

The system call rate is the rate at which system calls are being generated by the user processes. Every system call causes a switch to occur between user mode and system (or kernel) mode. A high system call rate typically corresponds to a high system CPU utilization.

If the system call rate is high, it is recommended to investigate which system calls are being generated, the frequency of each system call, and the average duration of each system call.

Context Switch Rate

This is the number of times the CPU switched processes (on average) per second. This is typically included in system CPU utilization, but some tools, like *glance*, track this separately.

Idle CPU

This is the percentage of time the CPU spent doing nothing (i.e. it did not execute any user or kernel code). It is good to see some, even lots, of idle CPU time. A non-idle CPU means the CPU run queue is never exhausted (or emptied), which means processes are always having to wait before reaching the CPU. The size of the line (CPU run queue) grows, as idle CPU time approaches 0.

CPU Run Queues/Load Average

Both these terms reference the same thing. This is the number of processes in the CPU run queue. For best performance, the average load in the CPU run queue should not exceed three.

5-11. SLIDE: CPU — Metrics to Monitor per Process

CPU — Metrics to Monitor per Process



- Process priority
- Process nice value
- Amount of CPU user time
- Amount of CPU system time

Student Notes

Individual processes vary greatly in terms of the load they place on the CPU. Metrics to monitor on an individual process include the following.

Process Priority

This is the priority of the process. If the priority is 127 or less, we know it is a real time process. If the priority is 128-177, either it is a system process, or it is a user process that is sleeping. If the priority is 178-255, then we know the process is executing in USER mode.

Process Nice Value

This is the **nice** value associated with the process. This only applies to time-share processes. This value determines how fast the process regains priority while it is waiting for the CPU. Small **nice** values (0-19) should be given to more important processes allowing them to regain priority quickly. Large **nice** values (21-39) should be given to less important processes, causing them to regain priority slowly.

User CPU Time vs. System CPU Time

This is the percentage of time the individual process spent in user mode (i.e. having the CPU execute user code) and system mode (i.e. having CPU execute kernel code). This is helpful in determining where the CPU spends its time when executing the process: user code or kernel code. It is generally desirable to see more time in user code.

5-12. SLIDE: Activities that Utilize the CPU

Activities that Utilize the CPU



- Process management
- File system I/O
- Memory management activities
- System calls
- Applications (for example, CAD-CAM and database processes)
- Batch jobs

Student Notes

Examples of activities that place a load on the CPU include the following.

System Activities

System activities are those activities which execute in kernel mode. Examples of system activities include system processes and user processes executing system calls.

- Process startup
- Process scheduling
- File system and raw I/O
- Memory management
- Handling of system calls

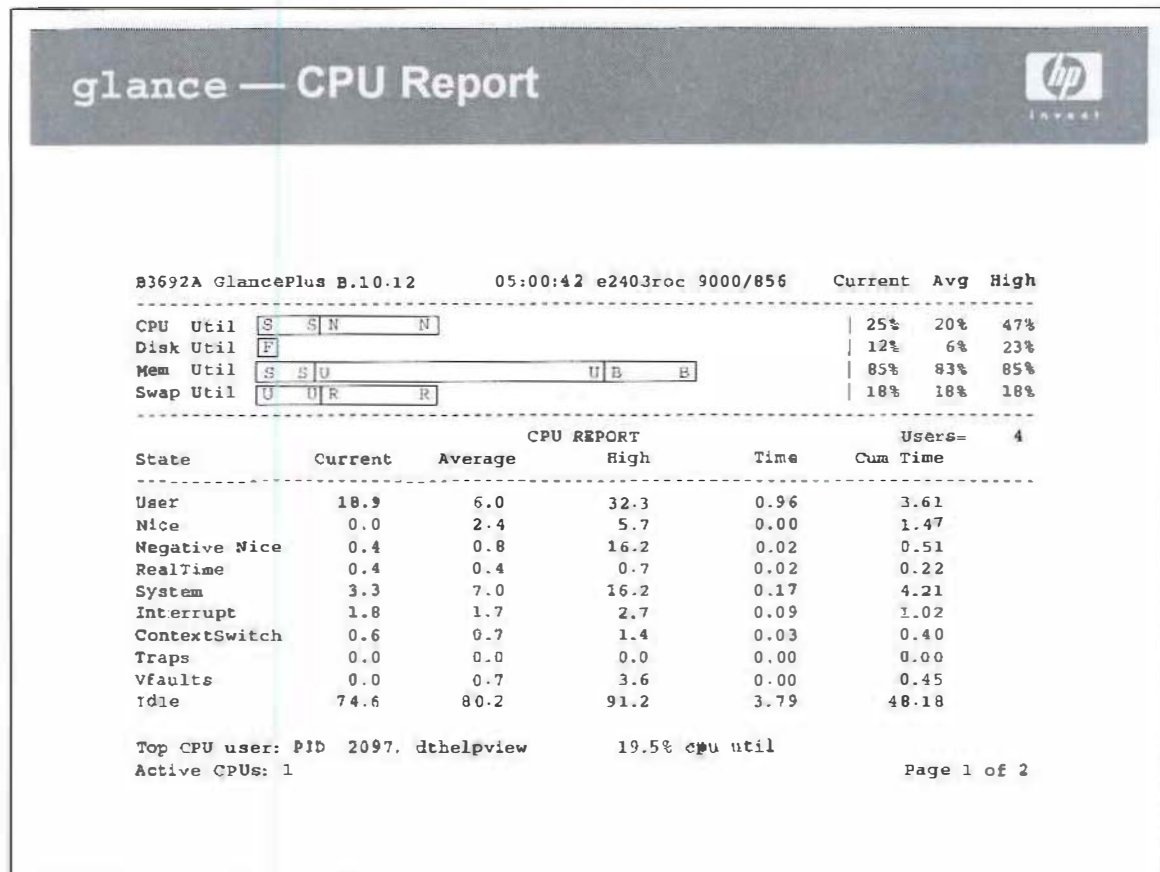
Module 5
CPU Management

User Activities

User activities are those activities that execute in user mode.

- CAD/CAM applications
- Database processing
- Client/server applications
- Compute-bound applications
- Background jobs (i.e. batch jobs)

5-13. SLIDE: glance — CPU Report

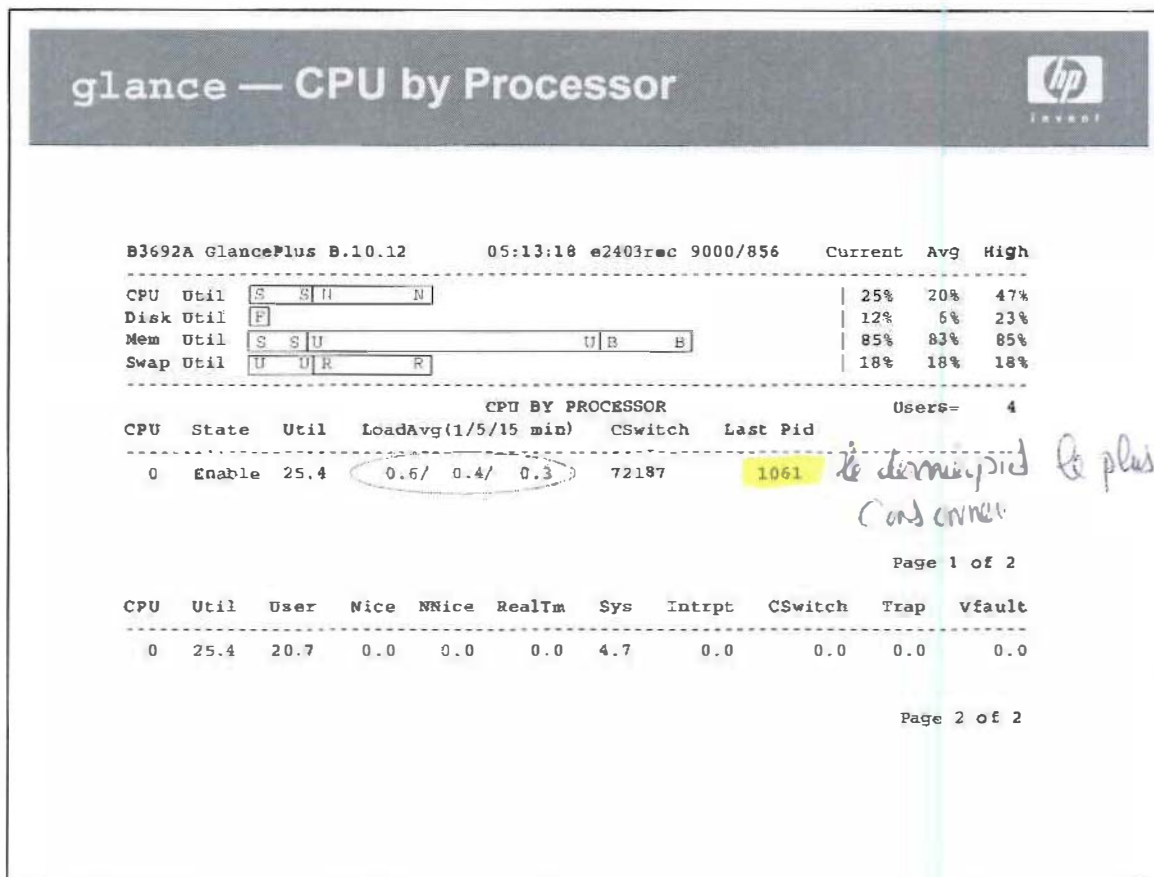


Student Notes

The **glance** CPU report (C key) provides details on where the CPU is spending its time from a global perspective.

- **User mode:** This is time spent by the CPU in user mode for all processes on the system. This includes processes with a nice value of 20 (user), processes with nice values between 21-39 (nice), processes with nice values between 0-19 (negative nice), and real-time priority processes.
- **System mode:** This is time spent by the CPU in system mode for all processes on the system. It includes time spent handling general system calls (system), and time spent handling interrupts, context switches, traps, and Vfaults (virtual faults).
- **Load Average:** This is the number of jobs in the CPU run queue averaged over three time intervals. It includes the average length of the run queue over the last 1 minute, the last 5 minutes, and the last 15 minutes. The CPU load average data is viewable on page 2 of this **glance** report. Also on page two are the System Call Rate, the Interrupt Rate, and the Context Switch Rate.

5-14. SLIDE: glance — CPU by Processor



Student Notes

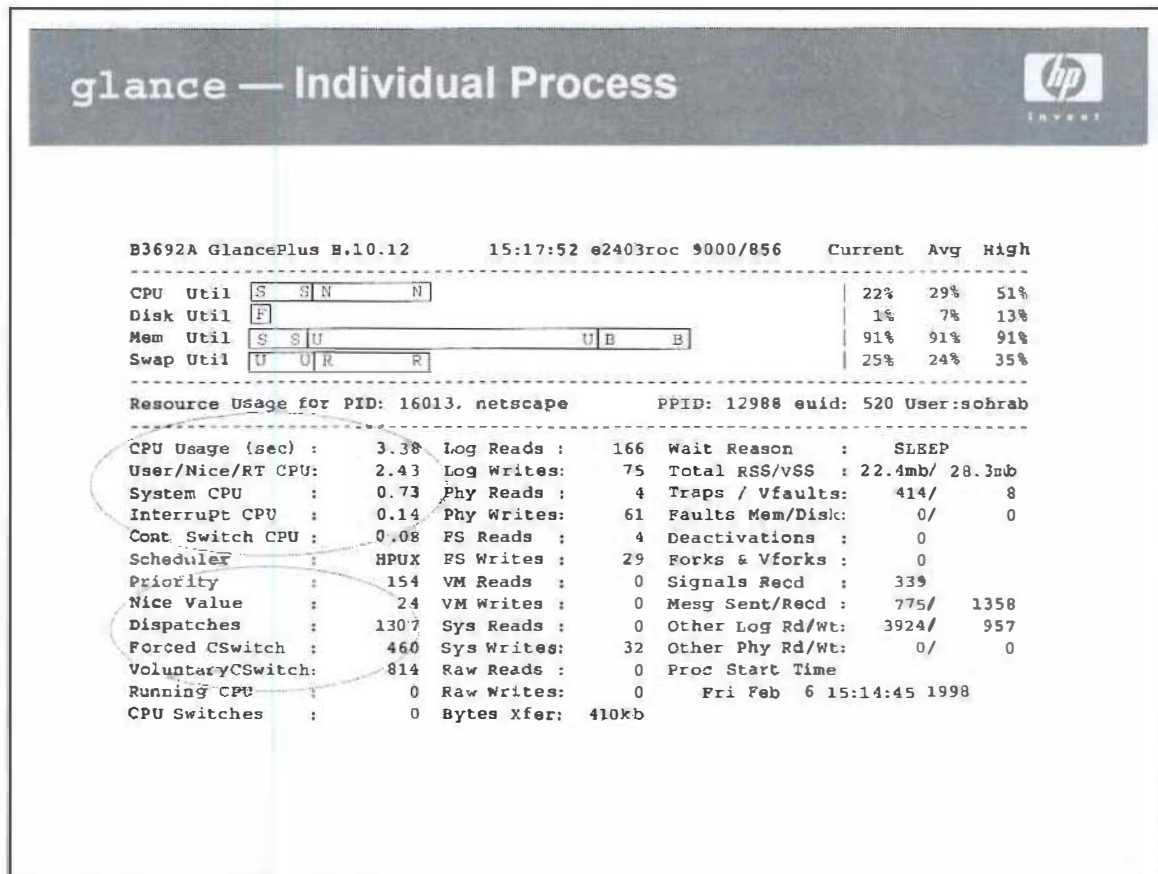
The glance CPU-by-processor report (a key) provides details on a per CPU basis.

CPU Utilization: This is the CPU utilization for the specific processor. If two or more processors exist on the system, the Global CPU Util bar graph shows an average CPU utilization. That is, a CPU that is 100% utilized and a second CPU that is 0% utilized will display 50% CPU utilization. This report displays utilization on a per processor basis.

Load Average: This is the number of processes, on average, in the CPU run queue over the last 1 minute, 5 minutes, and 15 minutes. This report displays CPU run queue information on a per processor basis.

Page two of this display shows the Utilization broken down into User mode, Nice, Negative Nice, Realtime, System, Interrupts, Context Switches, Trap and Virtual Faults on a per-processor basis.

5-15. SLIDE: glance — Individual Process

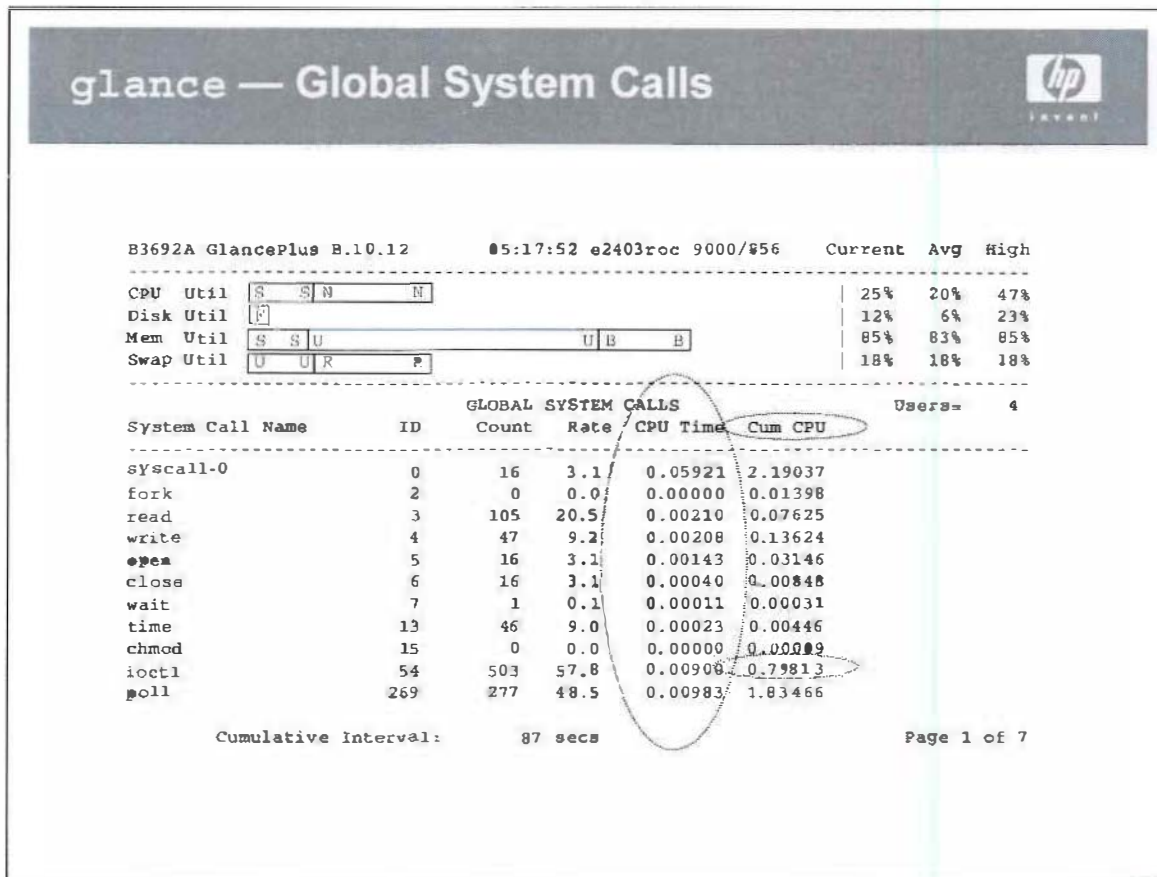


Student Notes

The **glance** individual process report (s key followed by the PID) displays CPU usage for an individual process, and the distribution of CPU time when executing the process (user, system, interrupt, context switch). Ideally, a process should spend more time in User/Nice/RT mode than in any of the other three modes.

Also displayed on a per-process basis is the Priority and Nice values for the selected process. In addition, the total number of forced context switches (time slice expiration or process preemptions) and voluntary context switches (process putting itself to sleep) are displayed.

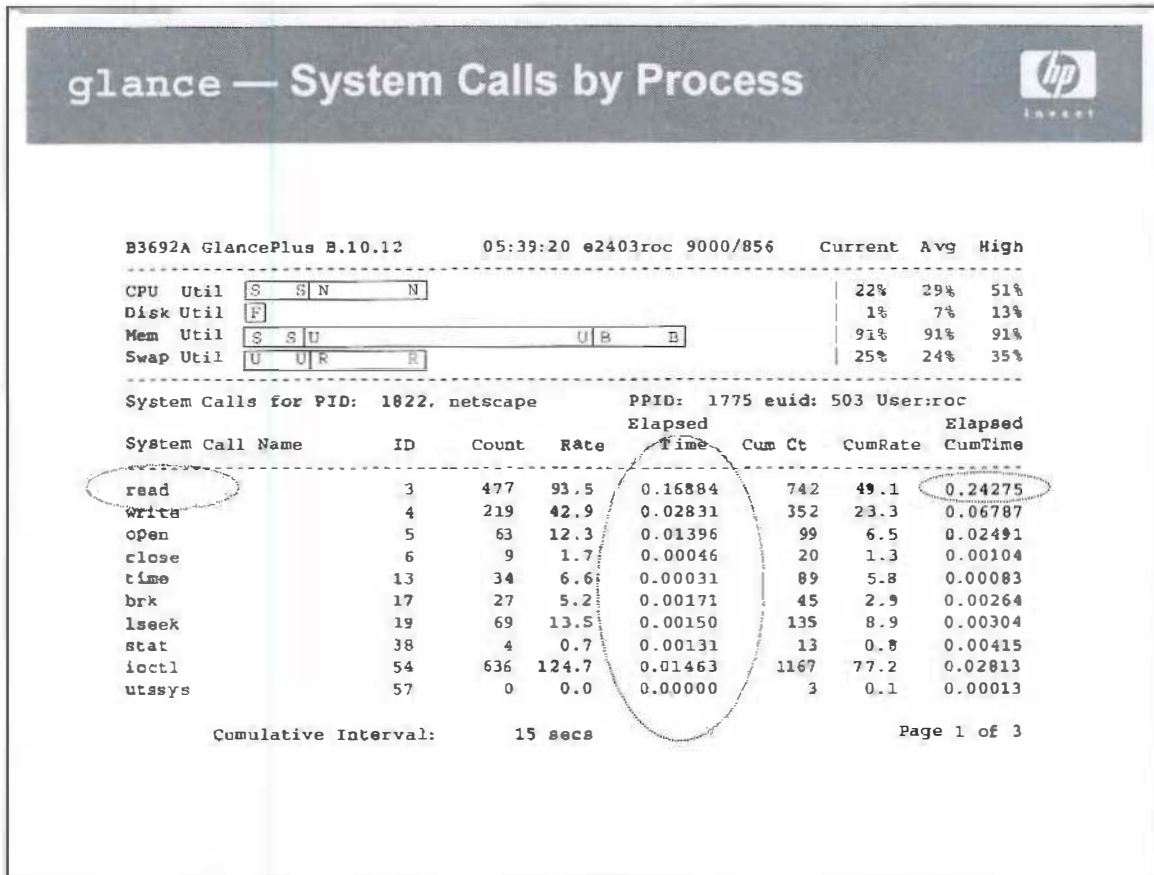
5-16. SLIDE: glance — Global System Calls



Student Notes

The **glance** global system calls report (Y key) displays all the system calls that have been executed system-wide. When system CPU utilization is high, this report can be used to identify on which system calls the CPU is spending most of its time.

5-17. SLIDE: glance — System Calls by Process



Student Notes

While examining an individual process, the system calls generated by that particular process can be viewed using the L key.

When the system time utilization is high for an individual process, this report can be used to view the specific system calls the process is performing, how many times the system calls are being invoked, and (most importantly) how much time is being spent by the CPU to execute the system calls.

The `read()` and `write()` system calls often take the most time, as they require physical I/O to the disk drives.

5-18. SLIDE: sar Command

sar Command



```
$ sar option <Interval size> <Number of intervals>
```

Options:

- u CPU Utilization (usr, sys, wio, idle)
- q Queue lengths/utilization (run, swap)
- M Above information in per-processor format
- c System calls

Student Notes

The **sar** command can be used to display global statistics on several important CPU operations. Using the **-u** option, information can be displayed on the time the system spent in User mode, System mode, Waiting for (disk) I/O, and idle. The Waiting for (disk) I/O is not reported by any other tool. Other tools simply lump it in with idle time. An example of the **sar** output with the **-u** option is shown below:

```
# sar -u 5 4
HP-UX r3w14 B.10.20 C 9000/712 10/14/97

08:32:24 %usr %sys %wio %idle
08:32:29 64 36 0 0
08:32:34 61 39 0 0
08:32:39 61 39 0 0
08:32:44 61 39 0 0

Average 61 39 0 0
```

Using the **-q** command, information can be displayed on the length and utilization of the run queue and the swap queue. We are most interested at this time in the run queue. An example of the **sar** output with the **-q** option is shown below:

```
# sar -q 5 4
HP-UX r3w14 B.10.20 C 9000/712 10/14/97

08:33:24      runq-sz %runocc swpq-sz %swpocc
08:33:29          8      100          0          0
08:33:34          8      100          0          0
08:33:39          8      100          0          0
08:33:44          8      100          0          0

Average          8      100          0          0
```

The **-M** option is always used in conjunction with **-u** and/or **-q**. It causes the metrics to be broken down by processor, so you can see how each processor is being utilized.

The **-c** option shows the total number of system calls being executed per second and singles out four specific system calls for further detail. They are the **read()**, **write()**, **fork()**, and **exec()** system calls. Also reported on this display is the average number of characters transferred in or out each second. An example of this output follows:

```
# sar -c 5 4
HP-UX r3w14 B.10.20 C 9000/712 10/14/97

08:33:24 scalls/s  sread/s  swrit./s  fork/s  exec/s  rchar/s  wchar/s
08:33:29      332         3         9      0.00      0.00   38630   2657
08:33:34      435         4        24      0.00      0.00   30310   2662
08:33:39      270         3        14      0.00      0.00    6758     0
08:33:44      524        20        15      0.20      0.20   73523     0

Average      390         7        15      0.05      0.05   37187   1331
```


5-19. SLIDE: `timex` Command

timex Command



```
$ timex prime_med

real      25.65
user      20.71
sys       3.43
```

Student Notes

The `timex` command can be used to benchmark how long the execution of a particular process takes in seconds. The command measures:

- **real** time — the amount of elapsed time from when the program started to when the program completed (sometimes referred to as the “wall clock” time).
- **user** time — the amount of time spent by the program executing in user mode.
- **sys** time — the amount of time spent by the program executing in kernel mode.

The example on the slide shows a total of 25.65 seconds elapsed from when the program `prime_med` started to when it completed. The execution spent 20.71 seconds executing in user mode and 3.43 seconds executing in kernel mode. The difference between **user + system** and **real** time is attributed to time the process spent *not* running on the CPU. The process may not get CPU time either because it was waiting on some resource (like disk or CPU) or because it was in a sleep state waiting for an event (like a child process waiting to finish executing).

5-20. SLIDE: Tuning a CPU-Bound System — Hardware Solutions

**Tuning a CPU-Bound System —
Hardware Solutions**



- Upgrade to a faster processor
- Upgrade the system with a larger data/instruction cache
- Add a processor to a multiprocessor system
- Spread applications to multiple systems

Student Notes

Practically speaking, the easiest performance gains are usually achieved by adding more and faster hardware. This could be upgrading to a faster processor, upgrading to a processor with more cache, adding another processor, or buying another system and off-loading some applications to the second system.

Upgrading to a faster processor may be possible with a simple module swap, but, more than likely, it would involve upgrading your entire system to a newer model. Some systems come with two or three possible processors and yours may not have the fastest available processors. If so, you may be able to upgrade the system's processors to faster versions without touching the rest of the system.

Nowadays, it's unlikely that you'll be able to upgrade the cache memory or TLB to larger sizes. Each processor chip seems to come with a predetermined amount of cache and Sized TLB. Only going to a different processor chip (and thus a larger model) will you be able to affect the cache memory and TLB sizes.

If your system is not yet at its full complement of processors, it may relieve your workload to add more processors. If you have a cell-based architecture, you may be able to add more processors to each cell, or even add more cells. Some servers come with extra processors

Module 5
CPU Management

installed, but not enabled. These systems have a feature called ICOD (Instant Capacity On Demand). By simply contacting HP, these disabled processors can be enabled, giving you more processing power with a minimum of time. If, at a later date, those processors are no longer needed, they can be disabled in a similar fashion.

Finally, if you have a system which is heavily loaded and another system which is lightly loaded, it may be possible to transfer some of the tasks from the busy system to the one which is less busy.

The disadvantage of these solutions is that most of them cost money.

5-21. SLIDE: Tuning a CPU-Bound System — Software Solutions

Tuning a CPU-Bound System — Software Solutions



- Nice less important processes
- Anti-nice more important processes
- Consider using `rtprio` or `rtsched` on most important processes
- Run batch jobs during non-peak hours
- Consider using PRM/MLM
- Consider using the processor affinity call `mpctl()`
- Optimize/recompile application

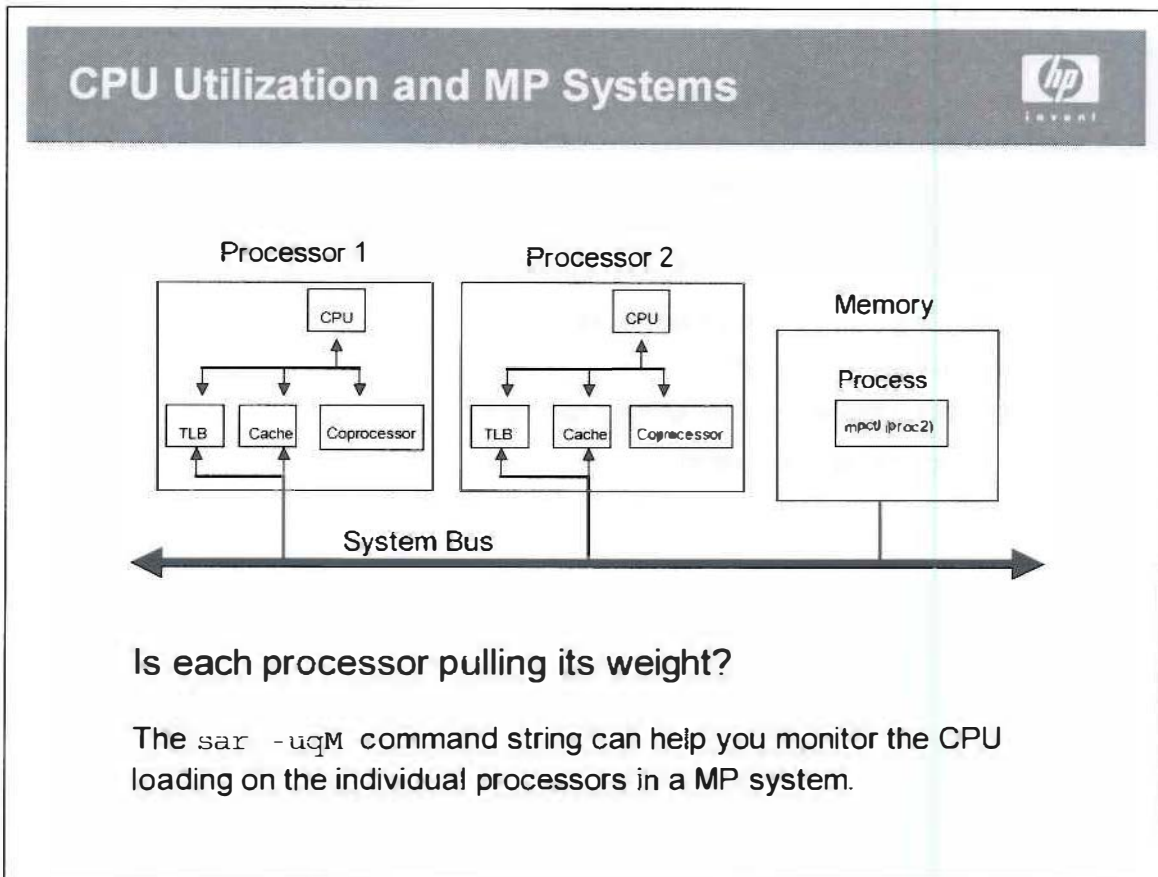
Student Notes

If the easiest performance gains are upgrading the hardware, then the greatest performance gains that are likely to be achieved are improving the software. A system with the fastest and most current hardware can still run slowly if the software is not configured properly.

One way to improve the performance of specific processes is to improve the priority of those processes. You can do this by improving the process's `nice` value or by making the process a real-time process. Or, you can reduce the `nice` value of other processes. Be careful when promoting a process to real time. If the process is not well-behaved, it can take over your entire system. By well-behaved, we mean that it is not compute bound and it is free of serious bugs.

Running batch jobs at non-peak hours has been a standard performance solution for many years on many systems. Other software performance improvements can be realized by using PRM (Process Resource Manager), WLM (Workload Manager), or the `mpctl()` system call.

5-22. SLIDE: CPU Utilization and MP Systems

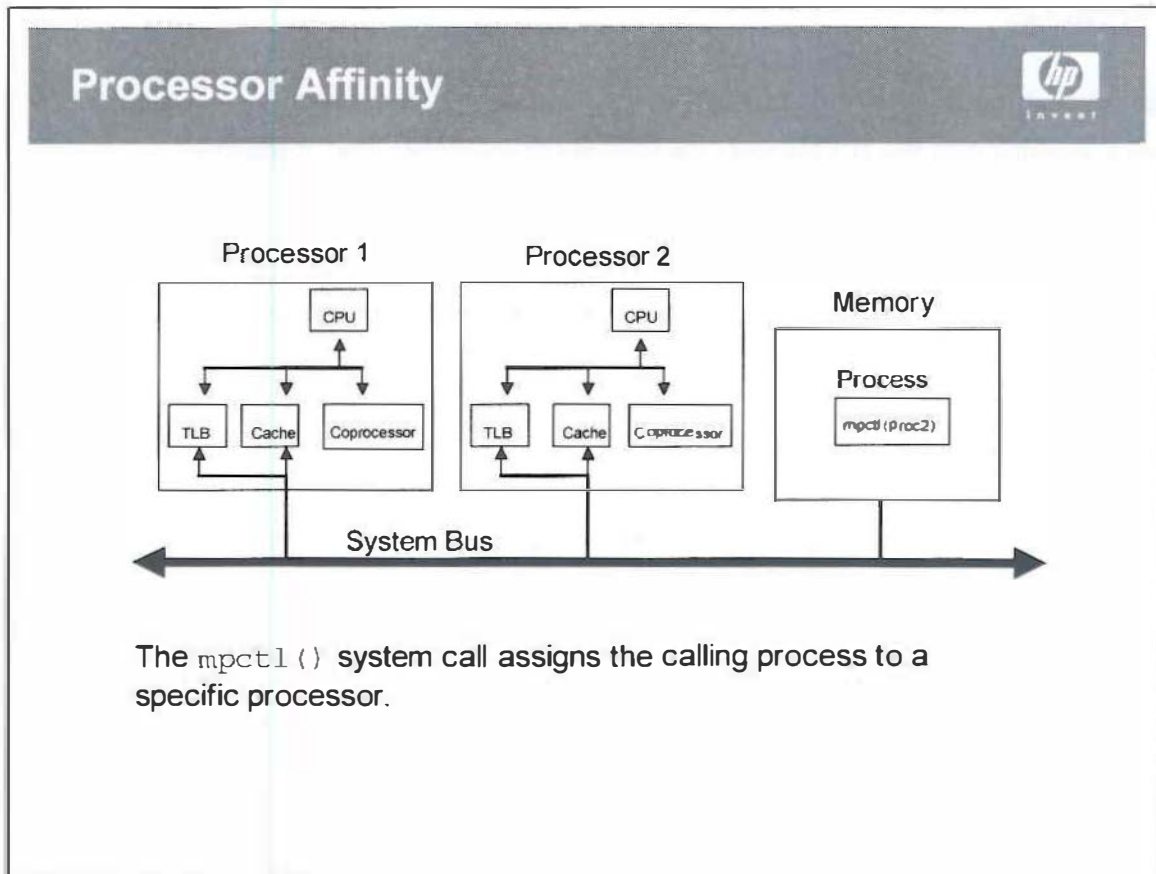


Student Notes

The `sar` command can be utilized to report CPU utilization for the overall system on a per-processor basis (when the `-u` and `-M` options are specified). In addition the `-q` option will report average run queue length while occupied, and percent of time occupied. Both of these metrics can assist in the evaluation of CPU loading and should be considered before making processor affinity calls.

`top` can also show you how your CPU resource is being distributed over the system. It automatically breaks down the load and utilization percentages on a per-processor basis when invoked. Remember, when you are running a system that supports Partitions (NPar or VPar), these tools only show you what is happening within a partition, as each partition has booted its own copy of the operating system and is acting as an independent system.

5-23. SLIDE: Processor Affinity



Student Notes

The `mpctl()` system call provides a means for determining how many processors are installed in the system (or partition), how many processors are in this pset, and assigning processes or threads to run on specific processors (also known as processor affinity) or within specific psets, and much, much more. Refer to the man page for `mpctl()` on your system.

Much of the functionality of this capability is highly dependent on the underlying hardware. An application that uses this system call should not be expected to be portable across architectures or implementations.

Processor sets are supported by the `pset()` system call. If your version of the operating system supports psets, refer to the man page for `pset()` for full details.

5-24. LAB: CPU Utilization, System Calls, and Context Switches

Directions

General Setup

Create a working data file in a separate file system (on a separate disk, if possible).

If another disk is available:

```
# vgdisplay -v | grep Name (Note which disks are already in use by LVM)
# iocan -fnC disk (Note any disks not mentioned above, select one)
# pvcreate -f <raw disk device file>
# vgextend vg00 <block disk device file>
```

In either case:

```
# lvcreate -n vxfs vg00
# lvextend -L 1024 /dev/vg00/vxfs <block disk device file>
# newfs -F vxfs /dev/vg00/rvxfs
# mkdir /vxfs
# mount /dev/vg00/vxfs /vxfs
# prealloc /vxfs/file <75% of main memory in bytes>
```

The lab programs are under `/home/h4262/cpu/lab0`

```
# cd /home/h4262/cpu/lab0
```

The tests should be run on an otherwise idle system — otherwise results are unpredictable. If the executables are missing, generate them by typing:

```
# make all
```

CPU Utilization: System Call Overhead

Use the **dd** command to size the read and write operations. Thus their number can be varied to change the number of system calls used to transfer the same amount of information. Then we can see the overhead of the system call interface.

The first command loads the entire file into buffer cache.

```
# timex dd if=/stand/vmunix of=/dev/null bs=64k
```

Now we take our measurements.

```
# timex dd if=/stand/vmunix of=/dev/null bs=64k
```

```
real 0 user 6 system 0
```



```
# timex dd if=/stand/vmunix of=/dev/null bs=2k
real 0.2 user 10.0 system 0.2

# timex dd if=/stand/vmunix of=/dev/null bs=64
real 17.0 user 0.9 system 5.4
```

System Calls and Context Switches

This lab shows you the maximum system call and context switch rates that your system can take. Three programs are supplied:

- **syscall** loads the system with system calls of one type
- **filestress** (shell script) generates file system-related system calls
- **cs** loads the system with context switches

1. What is the system call rate when your system is "idle"? 400
2. Run **filestress** in the background. What is the system call rate now? What system calls are generated by **filestress**? Take an average with **sar** over about 40 seconds i.e.

```
# sar -c 10 4
```

3. Terminate the **filestress** process by entering the following commands:

```
# kill $(ps -el | grep find | cut -c24-28)
# kill $(ps -el | grep find | cut -c18-22)
```

4. Run the **syscall** program and again answer question 2. Is the system call rate lower or higher than with **filestress**? Why?

Kill the **syscall** program, before proceeding.

```
# kill $(ps -el | grep syscall | cut -c18-22)
```

5. Using **cs**, compare the number of context switches on an idle system and a loaded system.

Idle _____ Loaded _____

6. Kill the **cs** program, remove the **/vxfs/file**, and dismount the **/vxfs** filesystem.

```
# kill $(ps -el | grep cs | cut -c18-22)
# rm -f /vxfs/file
# umount /vxfs
```

5-25. LAB: Identifying CPU Bottlenecks

Directions

The following labs are designed to show the symptoms of a CPU bottleneck.

Lab 1

1. Change directory to `/home/h4262/cpu/lab1`

```
# cd /home/h4262/cpu/lab1
```

2. Start the processes running in the background.

```
# ./RUN
```

3. Start a **glance** session and answer the following questions.

What is the CPU utilization? _____

What are the **nice** values of the processes receiving the most CPU time? _____

What is the average number of jobs in the CPU run queue? _____

4. Characterize the 8 lab processes that are running (**proc1-8**). Which are CPU hogs? Memory hogs? Disk I/O hogs etc. Identify processes that you think are in pairs.

5. Determine the impact of this load on user processes. Time how long it takes for the **short** baseline to execute.

```
# timex /home/h4262/baseline/short &
```

How long did the program take to execute? _____

6. Compare your results to the baseline established in the lab exercise in module 1, step 5.

7. End the CPU load by executing the **KILLIT** script.

```
# ./KILLIT
```

Lab 2

1. Change directory to `/home/h4262/cpu/lab2`.

```
# cd /home/h4262/cpu/lab2
```

2. Start the processes running in the background.

```
# ./RUN
```

3. In one terminal window, start `glance`.

In a second terminal window run

```
# sar -u 5 200.
```

Answer the following questions:

What does `glance` report for CPU utilization? _____

What does `sar` report for CPU utilization? _____

What is the priority of the process receiving the most CPU time? _____

How much time is the process spending in the `sigpause` system call? _____

How is the process being context switched (forced or voluntary)? _____

4. Determine the impact of this load on user processes. Time how long it takes for the `short` baseline to execute.

```
# timex /home/h4262/baseline/short &
```

How long did the program take to execute? _____

5. End the CPU load by executing the `KILLIT` script.

```
# ./KILLIT
```

Module 5
CPU Management

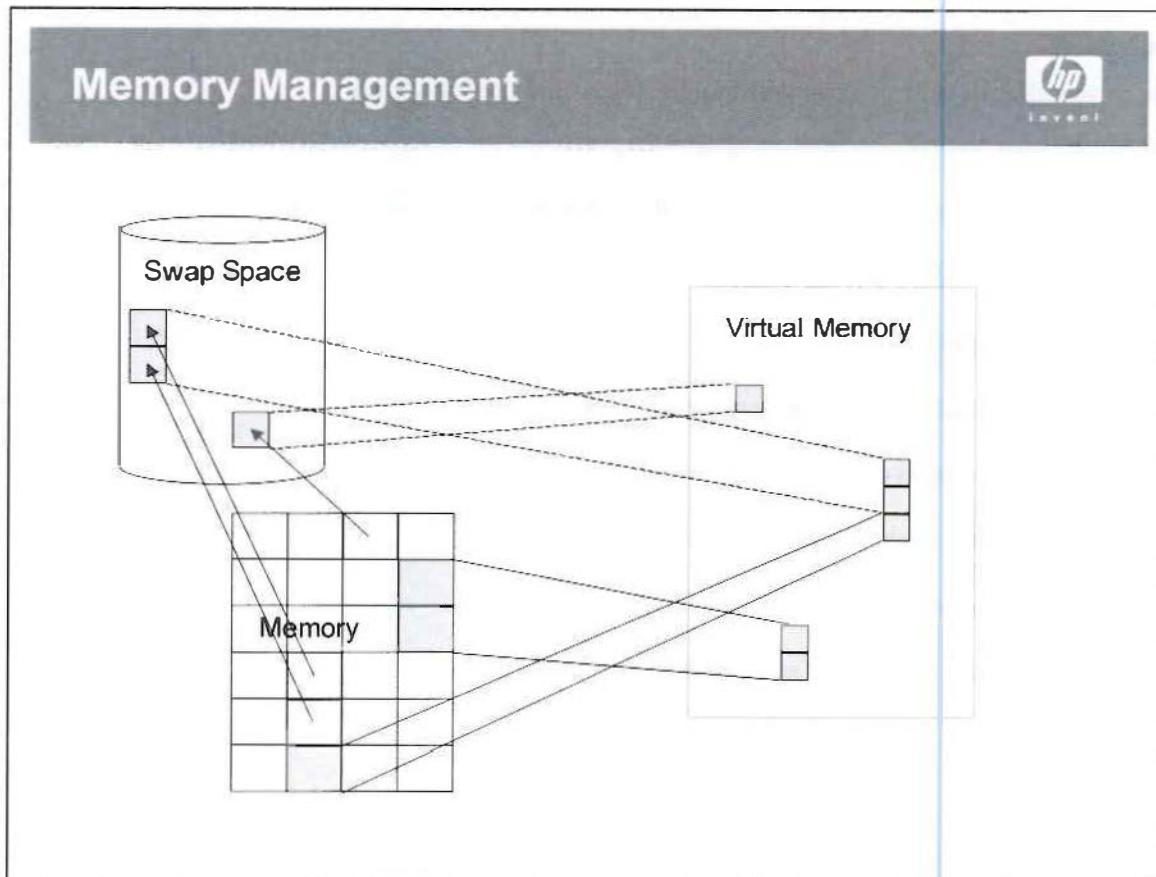
Module 6 — Memory Management

Objectives

Upon completion of this module, you will be able to do the following:

- Describe how the HP-UX operating system performs memory management.
- Describe the main performance issues that involve memory management.
- Describe the UNIX buffer cache.
- Describe the `sync` process.
- Identify the symptoms of a memory bottleneck.
- Identify global and process memory metrics.
- Use performance tools to diagnose memory problems.
- Specify appropriate corrections for memory bottlenecks.
- Describe the function of the `serialize` command.

6-1. SLIDE: Memory Management



Student Notes

Memory management refers to the subsystem within the kernel that is responsible for managing the main memory (also known as RAM) of the computer.

When managing main memory, the kernel allocates **memory pages** (default size is 4 KB) to processes as they need space. When main memory runs low on free space, the kernel will try to free up some pages in memory by copying those pages out to swap space on disk. The swap space can be thought of as an extension of main memory (like an overflow area) that is used when main memory becomes full. Processes paged out to the swap area cannot be referenced again until they are paged back in to main memory.

The term **virtual memory** refers to how much memory the kernel perceives as being available for allocation to processes. When the kernel allocates space to a process, it must track that page for the life of the process. Virtual memory includes main memory and swap space, as pages allocated to processes may be moved to swap space.

Example

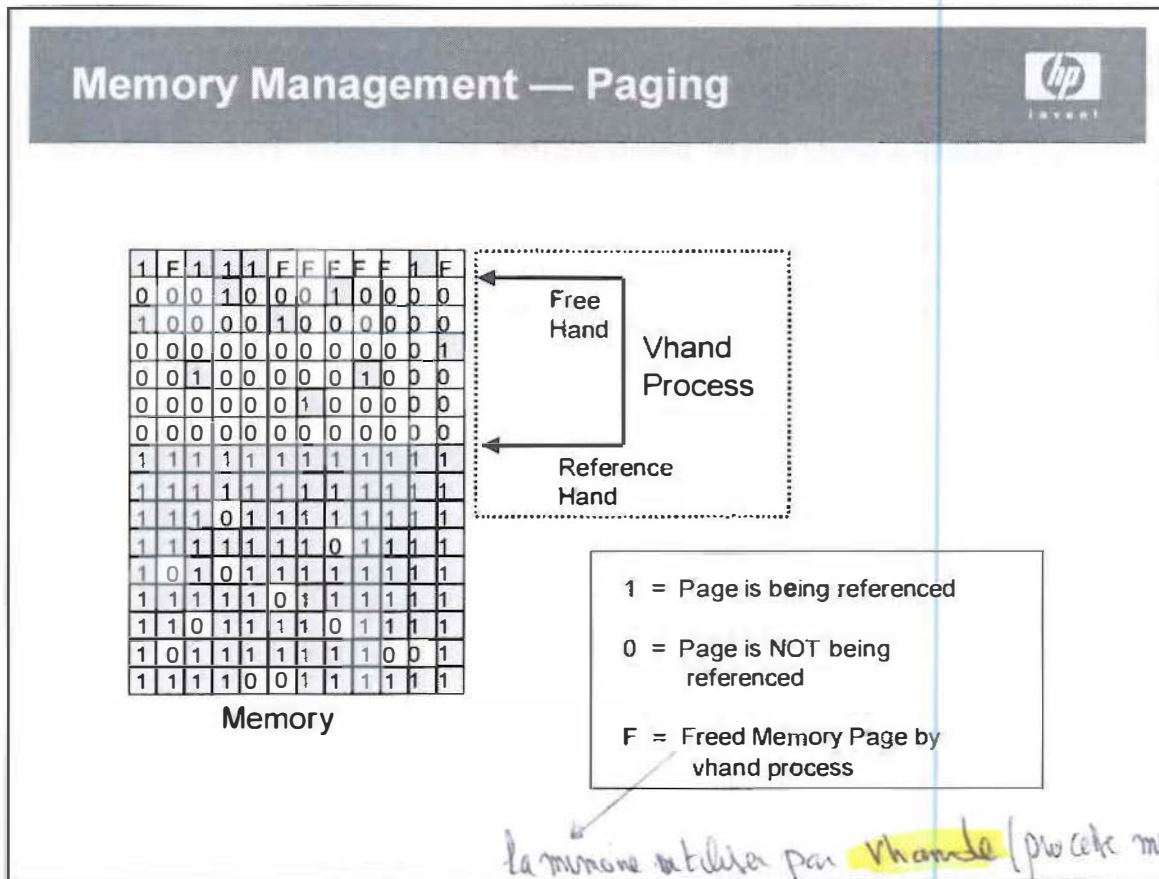
In the slide, there are three different processes being tracked: a one-page process, a two-page process, and a three-page process. The one-page process started in main memory and was

subsequently paged to swap space. The two-page process is entirely resident in main memory. And the three-page process has been partially paged to swap space (two of three pages are on swap).

From a virtual memory standpoint, the three processes are taking up six pages of memory: three pages in main memory and three pages on swap.

The preceding example is pretty simple. Reality is a little more complex. Processes actually consist of two basic types of pages, text and data. The data pages have “write” capabilities and thus their contents must be preserved when they are moved out of memory (to swap space). The text pages cannot be modified by the executing program. They are initially read in from the file system. If the memory manager should want to release the space that a text page is taking, it does not have to copy it out to swap, or even back to the file system.

6-2. SLIDE: Memory Management — Paging



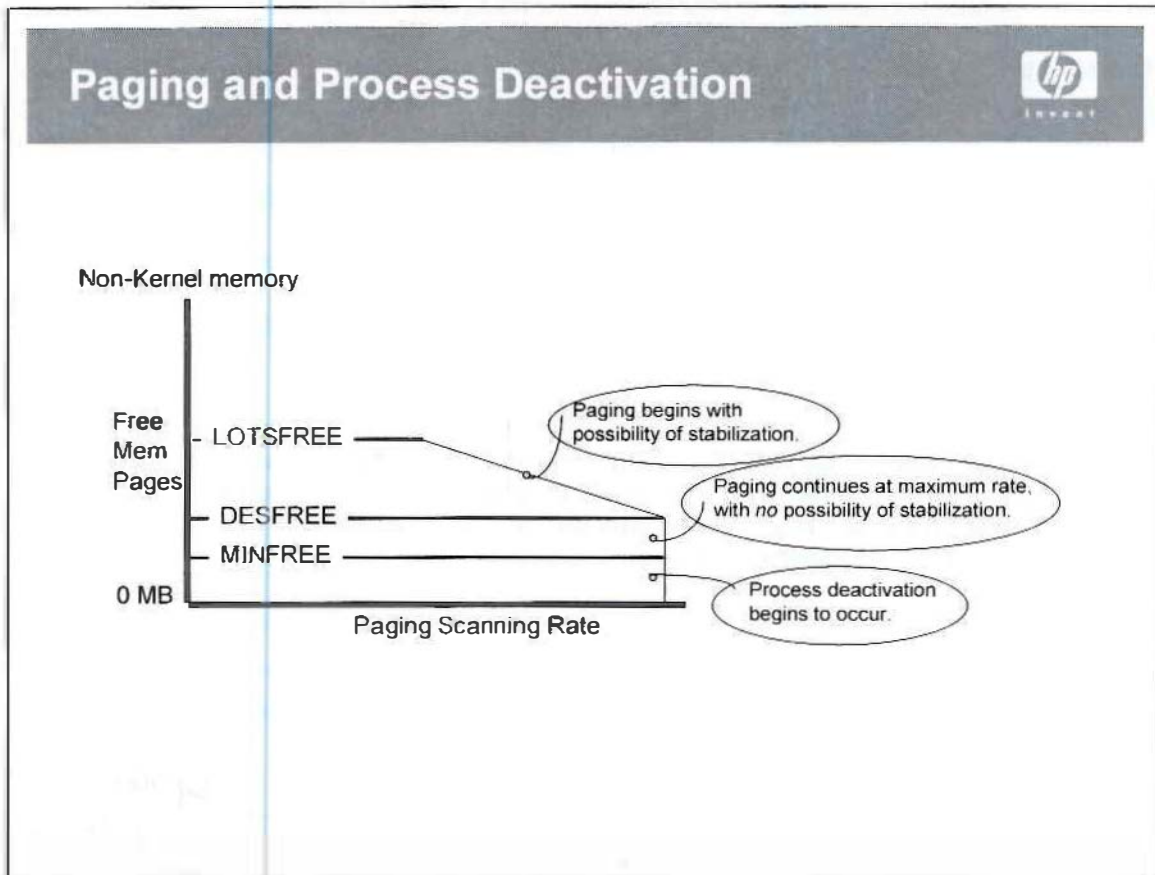
Student Notes

The **vhand** daemon is responsible for keeping a minimum amount of memory free on the system at all times. The **vhand** daemon does this by monitoring free pages and trying to keep their number above a threshold to ensure sufficient memory for efficient demand paging.

The **vhand** daemon utilizes a "two-handed" clock algorithm as seen on the slide. The first hand (also known as the "reference" hand or "age" hand) clears reference bits on a group of pages in an active part of memory. If the bits are still clear by the time the second hand (also known as the "free" hand or "steal" hand) reaches them, the pages are paged out.

The kernel automatically keeps an appropriate distance between the hands, based on the available paging bandwidth, the number of pages that need to be stolen, the number of pages already scheduled to be freed, and the frequency in which **vhand** runs. In essence, the distance between the hands determines how "aggressive" **vhand** is behaving. It behaves more aggressively as the memory pressure increases.

6-3. SLIDE: Paging and Process Deactivation



Student Notes

The system uses a combination of paging and deactivation to manage the amount of free memory. A minimum amount of free memory is needed to allow the demand paging system to work properly.

No paging occurs until the free memory falls below a threshold call LOTSFREE. Upon falling below LOTSFREE, paging will occur at a minimum level – becoming more aggressive as the number of free pages decreases. If the demand for memory continues, then paging will continue. However, if the demand for memory subsides, then there is a possibility that the amount of free memory will stabilize below the LOTSFREE threshold.

If free memory falls below a second threshold call DESFREE, then there is no possibility of stabilization (until free memory goes back above DESFREE) and the paging rate becomes much more aggressive compared to the initial paging rate.

Finally, if free memory falls below MINFREE, then process deactivation begins. A process is chosen by the kernel to be deactivated, and it is placed on the deactivation queue. Because the process is deactivated (therefore its pages are not being referenced) vhand will be able to page all its pages (including the uarea) out to the swap partition. The process will be

Module 6
Memory Management

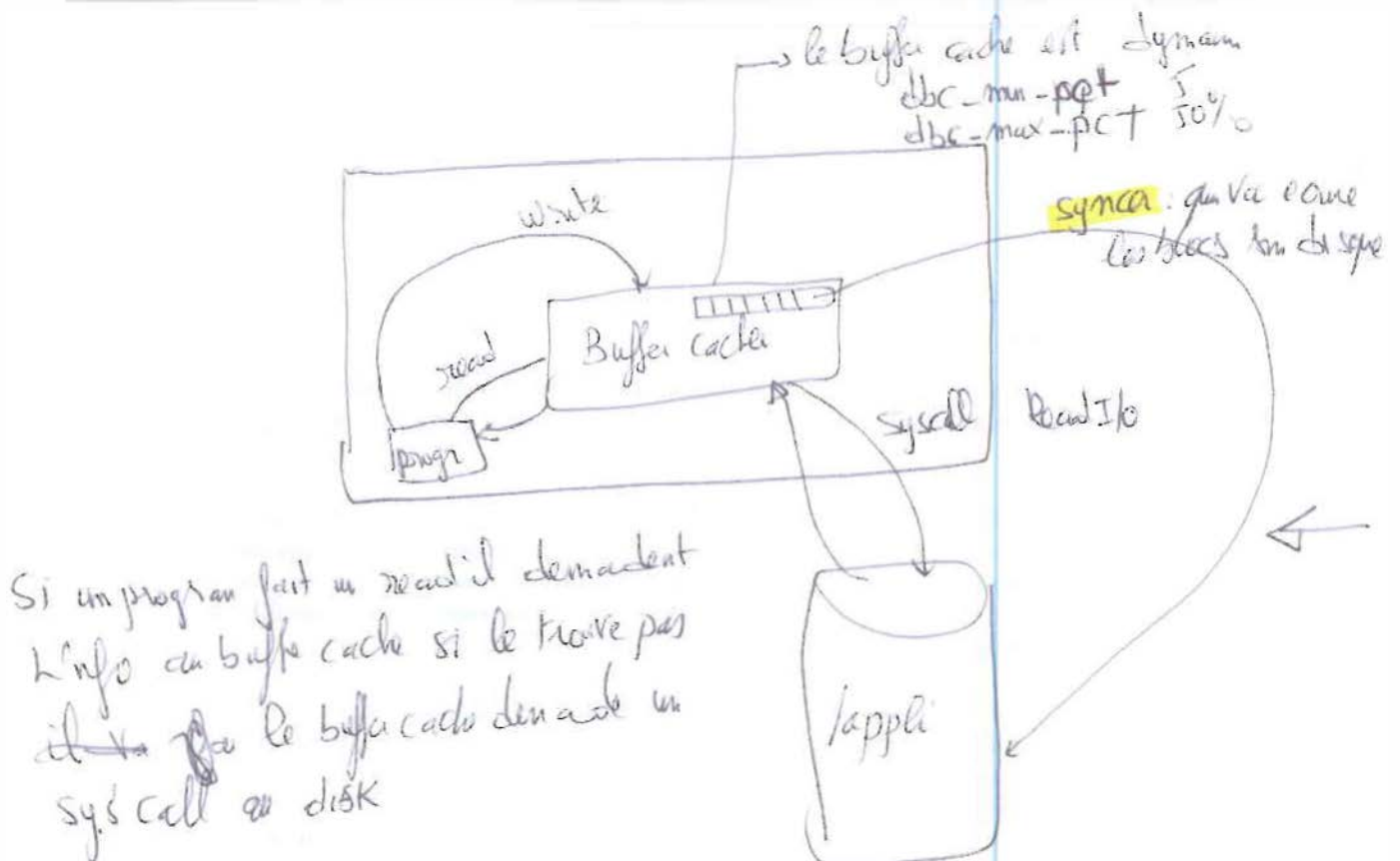
reactivated automatically once free memory rises above MINFREE. When a process is reactivated, only the uarea is immediately paged in. Other pages are faulted in as needed.

Below are the default formulae for LOTSFREE, DESFREE, and MINFREE.

(NKM = Non-Kernel Memory)

	$\leq 32 \text{ MB}$	$\geq 32 \text{ MB}, \leq 2 \text{ GB}$	$> 2 \text{ GB}$
LOTSFREE	1/8 of NKM	1/16 of NKM	64 MB
DESFREE	1/16 of NKM	1/64 of NKM	12 MB
MINFREE	1/2 of DESFREE	1/4 of DESFREE	5 MB

NOTE The values of LOTSFREE, DESFREE, and MINFREE were made tunable kernel parameters in HP-UX 11.00. Prior to the 11.00 release, these values were fixed and could not be changed. It is recommended by HP, however, that the parameters not be tuned manually.

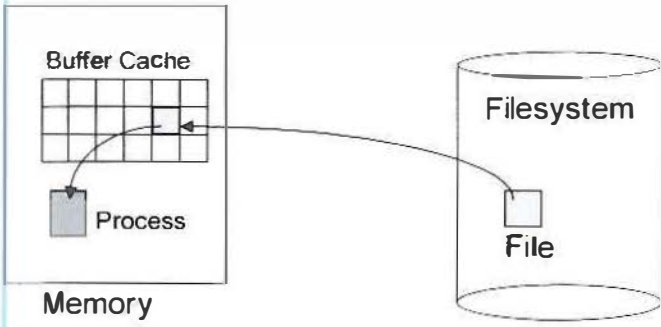


6-4. SLIDE: The Buffer Cache

The Buffer Cache



- Pool of memory designed to retain the most commonly accessed files from disk
- Used only for file system I/O (not raw I/O)
- Size of buffer cache controlled by `dbc_min_pct` and `dbc_max_pct`



Student Notes

The buffer cache exists to speed up file system I/O. The system tries to minimize disk access by going to disk as infrequently as possible, because disk access is often a bottleneck on most systems. Therefore, the most recently- or commonly-accessed files from disk persist in the portion of memory called the **buffer cache**.

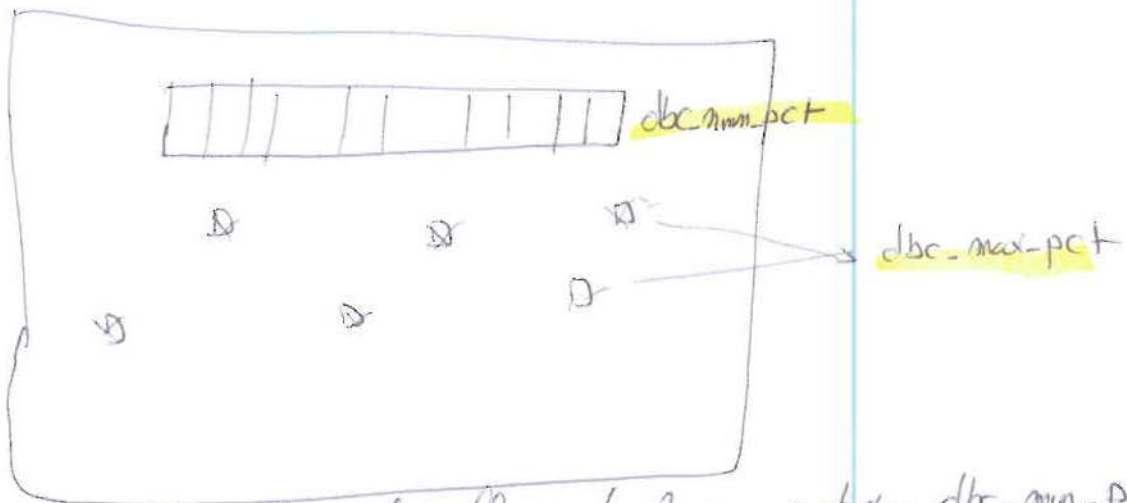
It is called dynamic because the size of the buffer cache grows or shrinks dynamically, depending on competing requests for system memory. Its minimum size is governed by the tunable parameter `dbc_min_pct`, and it cannot grow larger than the size specified in `dbc_max_pct`. These two parameters are expressed as percentages of total physical memory on the system.

Let's say `dbc_min_pct` is set to 10, while `dbc_max_pct` is 50. This means that initially 10% of physical memory is allocated to the buffer cache. As the system needs more space to buffer files read in from disk, the buffer cache will allocate more memory, and this will continue until it occupies 50% of memory, its maximum size. Later, when the system requires more memory for another use, say processes, the buffer cache could shrink an appropriate amount, but will never be less than the 10% minimum value. Therefore, a larger buffer cache is able to hold more files and will minimize their access time but will leave less memory available for other uses.

Module 6
Memory Management

NOTE: The buffer cache is dynamic in nature only when two other tunable parameters, **bufpages** and **nbuf**, are both set to their default values of 0.

Another example: if **dbc_min_pct** and **dbc_max_pct** are both set to the same value, say 20, the kernel will always use exactly that percentage of physical memory for the buffer cache.

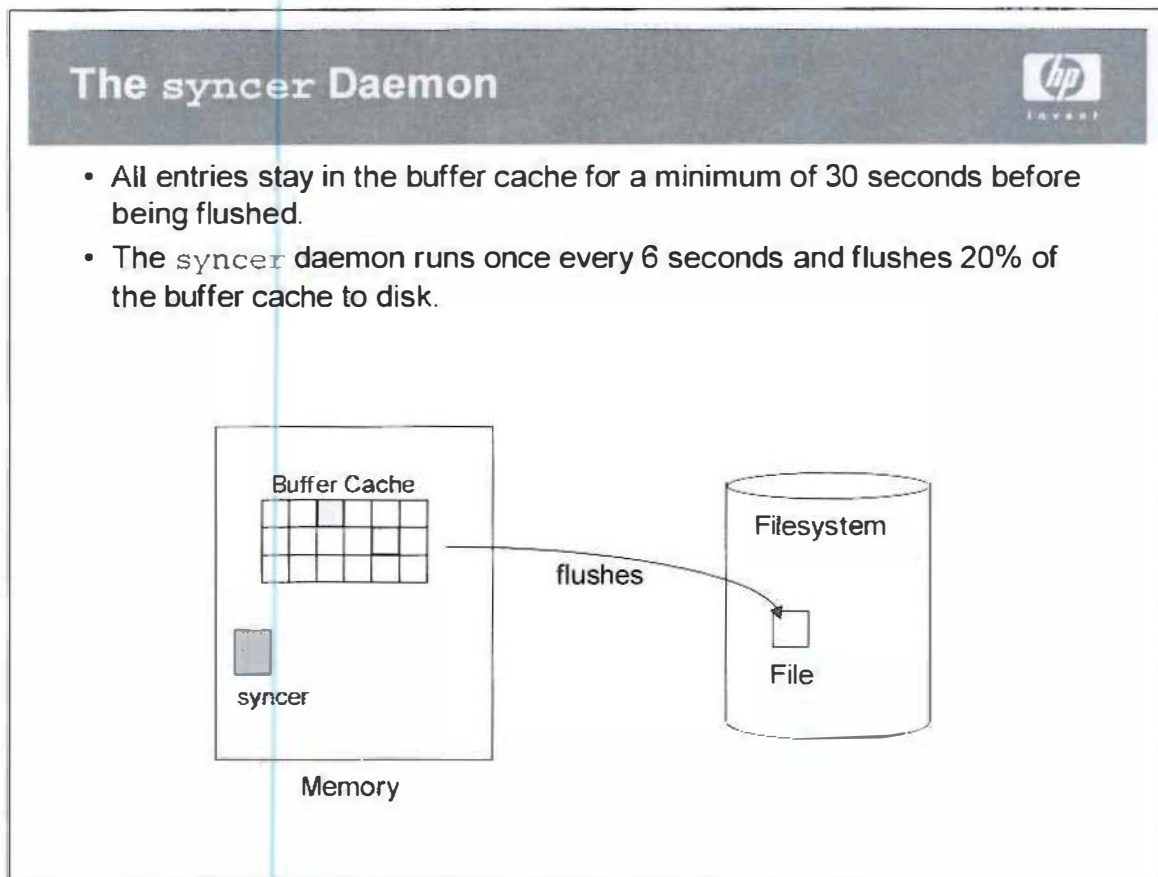


au bout de la machine le buffer cache alloue de façon continue **dbc_min_pct**

deux autres paramètres $\left(\frac{nbuf}{bufpage} \right) =$ priorité de kernel

mmode n'est pas utilisée maintenant il y'a le vxmmode

6-5. SLIDE: The syncer Daemon



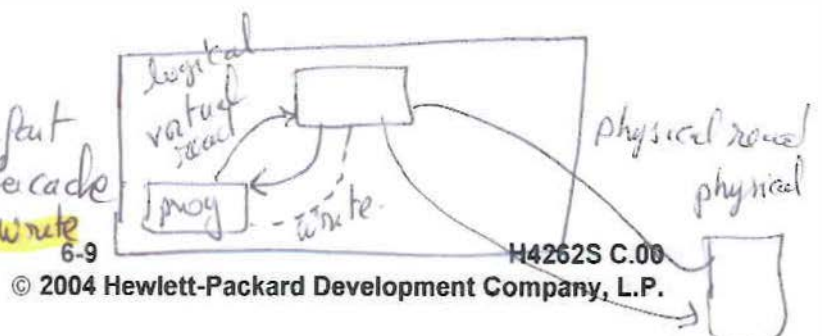
Student Notes

For disk writes, data flows from the buffer cache to disk. How does it get to the buffer cache? The kernel writes data to it. The `syncer` process takes care of flushing data in the buffer cache to the files on the disk.

When a user edits a file, makes changes to that file, and saves the changes, those changes do not go to disk right away. The kernel writes the data to the buffer cache, and some time later (within 60 seconds) the data finally arrives at the disk. This time period is chosen as a balance between ensuring that the file system is fairly up-to-date in case of a crash and efficiently performing disk I/O.

There are many applications that do not rely on the operating system's built-in processes to flush data to disk, but instead take over that operation themselves. In other words, they create their own buffers and manage the flushing at appropriate intervals. A common example is a database application that needs to guarantee the completion of a transaction within a specified time interval.

Si write de prog sur buffer cache
Si espace pas dans le buffer cache fait
un sync pour vide le buffer cache
il sera considéré comme **write**
<http://education.hp.com>
Si espace existe il sera considéré
comme logical write

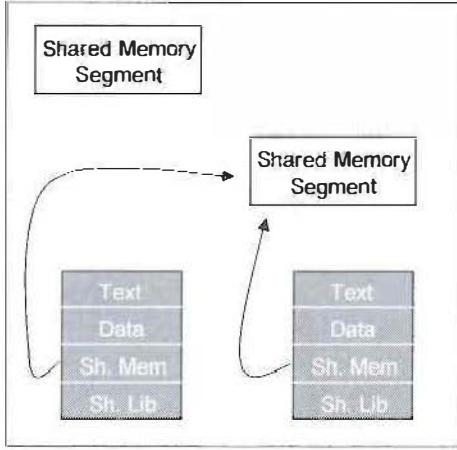


6-6. SLIDE: IPC Memory Allocation

IPC Memory Allocation



```
# ipcs -mob
IPC status from /dev/kmem as of Sat Feb 14 06:53:27 1998
T      ID      KEY      MODE      OWNER      GROUP      NATCH      SEGSZ
Shared Memory:
m      5      0x06347849--rw-rw-rw-   root      root       0          77384
m      7      0x000c0568 --rw-----   root      root       2         131516
```



Memory

Student Notes

UNIX implements interprocess communications using different mechanisms. Three mechanisms that require additional system memory are semaphores, shared memory, and message queues.

- **Semaphores** are used to synchronize memory resources between competing processes.
- **Shared memory** segments are resources capable of holding (in memory) large amounts of data that can be shared between processes.
- **Message queues** hold strings of information (messages) that can be transferred between processes. Two types of processes that utilize message queues are networking and database processes.

Shared memory provides a mechanism to reduce interprocess communication costs significantly. Two processes that are ready to share data, address the same portion of shared memory into their addressable space. Changes made to the shared memory are seen immediately by all processes and do not require kernel services. So from a kernel perspective, other than initially setting up the shared memory, there is very low cost in using shared memory.

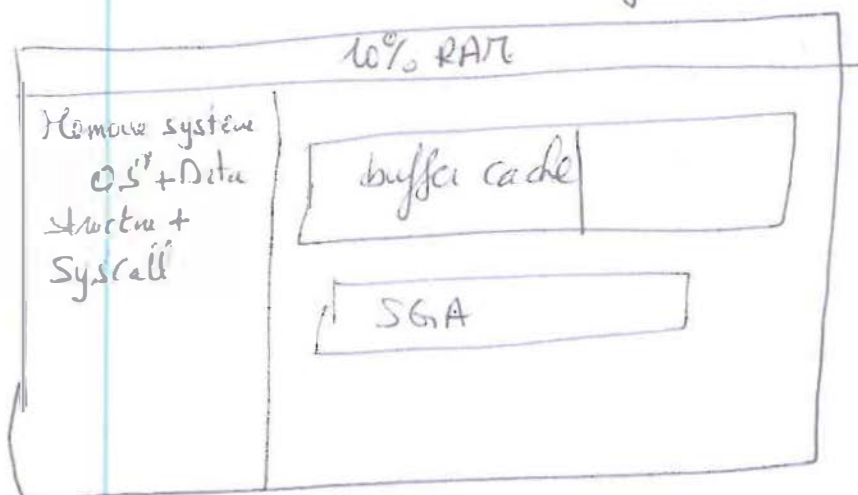
On the slide, each process has a shared memory segment that references one and the same shared memory area.

The more processes that allocate shared memory segments, the higher the memory usage. The shared memory segments in physical memory can be viewed with the `ipcs -mob` command or a reporting tool like `glance`. From time-to-time, they might have to be cleaned up or removed manually if an application terminates ungracefully. This is done by the `superuser` with the `ipcrm` command.

A worthwhile baseline measurement for a system administrator is to run the `ipcs -mob` command during a quiet period. It is also eye opening to repeat this command when the system is at its busiest.

`ipcrm 0x06347849`

Exemple : je veux allouer 500 Mo pour SGA d'oracle
pour 2 GB



114
RAM.

pour 266 B de RAM
c'est pas fixe par exemple

Res max = 20 Mo
system = 500 Mo
buffer cache = 50% 1024 Mo
SGA = 200 Mo
user = 100 Mo

Lockable memory = 10% de memoire + 114 RAM
grep -i physical /etc/passwd

6-7. SLIDE: Memory Metrics to Monitor — Systemwide

Memory Metrics to Monitor — Systemwide



- Is vhand Active? → *Scann R+ libérer plus pages et active*
 - Pages scanned by vhand (SR)
 - Pages freed by vhand (FR)
 - Pages paged out
- Is swapper Active?
 - Processes deactivated (SO)
 - Amount of free memory relative to
 - lotsfree
 - desfree
 - minfree
- Size of dynamic buffer cache
- Size of IPC Shared memory segments

si j'ai des desactivable memoire donc j'ai des gèrèl gèlles memoire

Student Notes

The utilization of memory can be monitored in a number of different ways. There are multiple tools and multiple metrics that monitor memory usage. The first metrics you want to look at are those that will tell you whether **vhand** is active.

Pages Scanned by vhand

This is the number of pages the **vhand** process has scanned (i.e. dereferenced with the reference hand) when looking for pages to free in memory. This tells you that **vhand** is actively scanning pages in an attempt to free them up. There is some memory pressure.

Pages Freed by vhand

This is the number of pages the **vhand** process has freed (i.e. the reference bit was still dereferenced when the free hand looked at it). The ratio between *pages scanned* and *pages freed* indicates how successful the **vhand** process is when looking for memory pages to free.

Amount of Paging

This indicates the level of disk activity to the swap partition. If a consistent amount of paging to swap space is occurring, then performance is impacted (most likely significantly).

Next, check to see if the **swapper** is active.

Process Deactivations

This indicates that processes are being deactivated, meaning free memory has fallen below the MINFREE threshold. There is severe memory pressure.

Amount of Free Memory

This indicates the severity of the free memory situation. If free memory has fallen below LOTSFREE, then we know some paging has taken place. **vhand** is active. If it is below **DESFREE**, then the situation is more severe, and much more paging is occurring. **vhand** is aggressively active. Finally, if free memory is below MINFREE, then a high level of paging ~~and~~ process deactivation is occurring. **vhand** and **swapper** are both active. To determine what the values are for lotsfree, desfree, and minfree, use the following commands:

```
# echo "lotsfree/D" | adb -k /stand/vmunix /dev/mem
# echo "desfree/D" | adb -k /stand/vmunix /dev/mem
# echo "minfree/D" | adb -k /stand/vmunix /dev/mem
```

The settings for these three values in the kernel will then be displayed in 4K pages. You can then compare them to the current size of the free page list. These values will not change, unless you change the size of Non-Kernel Memory. (Remember the formulas shown earlier?)

Size of Dynamic Buffer Cache

This is the amount of memory being consumed by the buffer cache. If memory is full and the buffer cache is large, it will most likely cause paging, since the buffer cache typically shrinks slower than the rate at which new memory is needed. Heavy disk I/O demands may prevent the buffer cache from shrinking at all.

Size of IPC Shared Memory Segments

This is the amount of memory used for interprocess communications. Of special interest will be the number and sizes of shared memory segments, as these can be quite large, especially if graphical applications or a database management system is being used.

6–8. SLIDE: Memory Metrics to Monitor — per Process

Memory Metrics to Monitor — per Process



- Size of RSS/VSS
- Size of text, data, and stack segments
- Number of shared memory segments
- Amount of time blocked on virtual memory

Student Notes

Individual processes vary greatly in terms of the amount of memory they use. Metrics to monitor memory utilization on a per-process basis include the following:

Size of RSS/VSS

The Resident Set Size (RSS) for a process is the portion of the process (in KB) that is currently resident in physical memory. Since the entire process does not have to be resident in memory in order to execute, this shows how much of the process is actually resident in memory.

The Virtual Set Size (VSS) for a process is the total size of the process (in KB). This indicates that if the entire process were to be loaded, this is how much memory the entire process would consume. Very rarely is the entire process resident in memory. If the entire process were in memory, then the RSS value would be equal to the VSS value.

Size of Text, Data, and Stack Segments

These are the RSS and VSS sizes for the three main components of a process. Since every process has a single text, data, and stack segment, these values should be monitored, especially for large processes. The data segment is the most likely to be large.

Each of these three segments has a maximum size to which they can grow – limited by tunable kernel parameters. They are `maxtsiz`, `maxdsiz`, and `maxssiz` for a 32-bit process. They are `maxtsiz_64bit`, `maxdsiz_64bit`, and `maxssiz_64bit` for a 64-bit process. If a process tries to grow one of these segments beyond its maximum size, then the process terminates (and in some cases “core dumps”).

Number and Size of Shared Memory Segments

These are the shared memory segments to which a process is attached. The maximum size of a shared memory segment is limited by the kernel parameter, `shmmax`. The number of shared memory segments a process can attach to is limited by the kernel parameter, `shmseg`.

Amount of Time Spent Blocked on Virtual Memory

This is the amount of time the process was prevented from executing because it was waiting (or blocked) on a text or data page to be paged in.

6-9. SLIDE: Memory Monitoring `vmstat` Output

Memory Monitoring <code>vmstat</code> Output												
												
#=> <code>vmstat -n 5</code>												
VM												
memory												
	avm	free	re	at	pi	po	fr	de	sr	in	sy	cs
	9140	3824	3	4	0	0	0	0	0	675	824	140
CPU												
cpu												
us	sy	id	r	b	w							
9	5	86	1	100	0							
9017			3500	41	49	11	0	0	0	1257	2823	329
24	17	60	0	100	0	41	0	0	0	1419	3795	481
10292			2255	65	20	41	0	0	0	1419	3795	481
67	24	9	5	102	0	85	0	0	0	1698	4771	641
10227			976	89	19	85	0	0	0	1698	4771	641
67	33	0	7	103	0	91	48	26	0	194	1791	5847
10958			400	81	12	91	48	26	0	194	1791	5847
67	31	3	8	110	0	98	51	24	0	268	1598	4313
10759			454	33	3	98	51	24	0	268	1598	4313
62	20	18	6	111	0	65	74	39	0	282	1021	3175
13448			404	21	0	65	74	39	0	282	1021	3175
32	15	53	0	118	0							

Student Notes

A useful command to view virtual memory statistics is `vmstat`. The slide shows `vmstat`'s output being updated every 5 seconds.

When viewing `vmstat`'s output, always keep an eye on the `po` (pages paged out) parameter. Ideally, you want this to be zero, indicating no paging out is occurring.

Statistics regarding the `vhand` algorithm, the `fr` (pages freed per second) and `sr` (pages scanned by the clock algorithm, per second) parameters show the actual behavior of `vhand`.

Output Headings

`procs`

- `r` In run queue
- `b` Blocked for resources (I/O, paging, and so on)
- `w` Runnable or short sleeper (less than 20 seconds) but deactivated

memory

avm	Active virtual pages (run during the last 20 seconds)
free	Size of free list page (in 4K pages)
re	Page reclaims per second
at	Address translation faults per second (page faults)
pi	Pages paged in per second
po	Pages paged out per second
fr	Pages freed per second
defacto	Anticipated short term memory shortfall
sr	Pages scanned by algorithm per second

faults

in	Non-clock device interrupts per second
sy	System calls per second
cs	CPU context switches per second

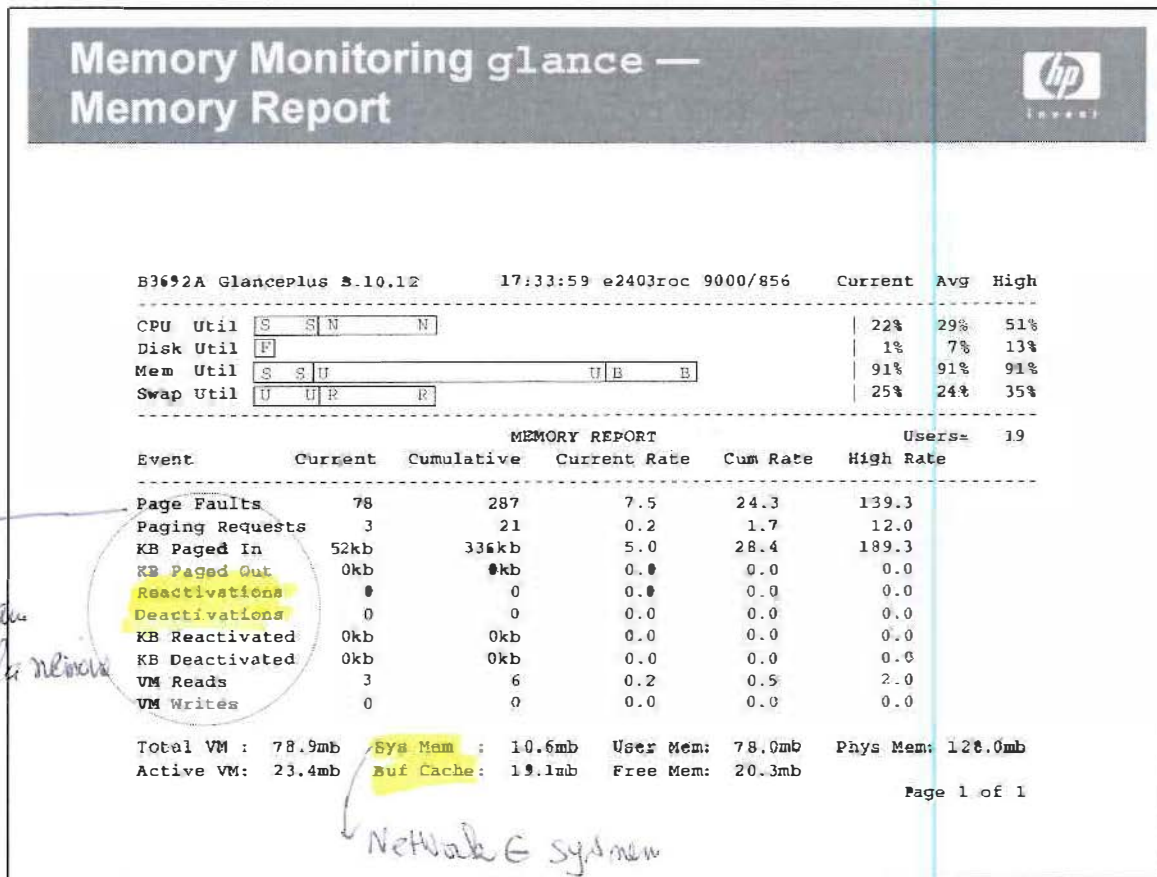
CPU

us	Percentage of time CPU spent in user mode
sy	Percentage of time CPU spent in system mode
id	Percentage of time CPU is idle

with -S option

si	Processes reactivated per second
so	Processes deactivated per second

6-10. SLIDE: Memory Monitoring glance — Memory Report

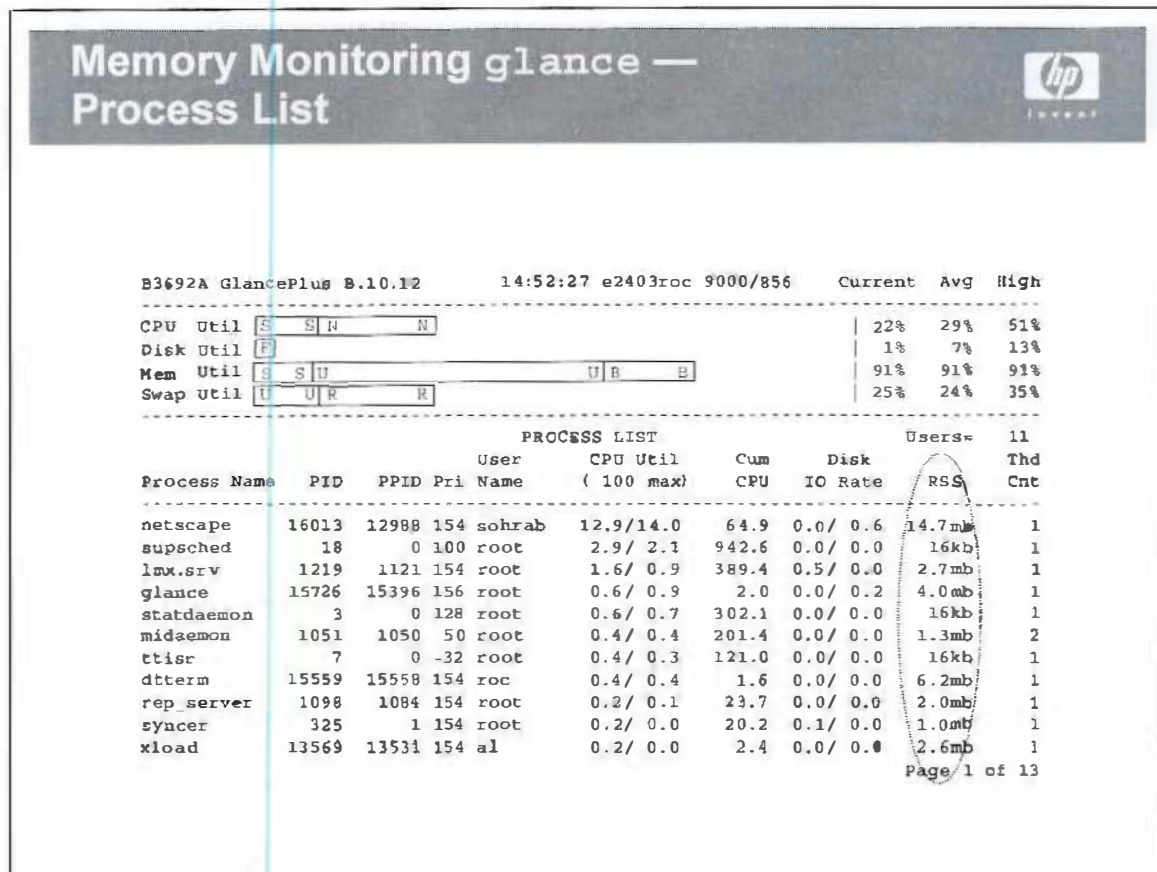


Student Notes

glance has extensive memory monitoring abilities. Like **vmstat**, it can give paging statistics, in addition to showing if any processes are being deactivated. Remember, this is an indication of severe memory shortage.

There is other valuable information on this report, such as the statistics at the bottom showing the current **Dynamic Buffer Cache** size, the current amount of **Free Memory**, and the total **Physical Memory** in the system.

6-11. SLIDE: Memory Monitoring glance — Process List



Student Notes

The **glance** Process List report can be used to monitor process statistics, including how much memory processes are currently consuming. The highlighted column, **RSS** (Resident Set Size), shows memory being used on a per-process basis. Very simply put, this helps to identify the "memory hogs" on the system.

For example, the process called **netscape** has an RSS of 14.7 MB, while **statdaemon** is minimal. Other large processes include **glance**, **xload**, and **dtterm**. What do all these processes have in common? They are all GUI (graphical user interface) programs running as windows in a graphical window environment.

Moral: programs that open their own windows are relatively memory-intensive and should be minimized. Users should be encouraged not to leave several windows open on their screens if they do not have a continuing need for them.

6-12. SLIDE: Memory Monitoring glance — Individual Process

Memory Monitoring glance — Individual Process




```

B3692A GlancePlus @.03.70.00      15:52:03  r206c42 9000/800      Current Avg High
-----
CPU Util  S  S N N                      | 15% 15% 15%
Disk Util  F                               | 1% 0% 2%
Mem Util   S S U U B B                  | 96% 96% 96%
Swap Util  U U R R                      | 15% 15% 15%
-----
Resources PID: 28030, glance          PPID: 27993 euid: 0 User: root
-----
CPU Usage (util):      0.1 Log Reads :      1 Wait-Reason : STOPS
User/Nice/RT CPU:     0.1 Log Writes:      0 Total RSS/VSS : 3.6mb/ 5.6mb
System CPU           : 0.0 Phy Reads :      0 Traps / Vfaults: 1/ 10
Interrupt CPU        : 0.0 Phy Writes:      0 Faults Mem/Disk: 6/ 0
Cont Switch CPU      : 0.0 FS Reads :      0 Deactivations : 0
Scheduler            : HPUX FS-Writes:      0 Forks & Vforks : 0
Priority              : 154 VM Reads :      0 Signals Recd : 0
Nice Value           : 10 VM Writes :      0 Msg Sent/Recd : 0/ 0
Dispatches           : 6 Sys Reads :      0 Other Log Rd/wt: 32/ 172
Forced CSwitch       : 0 Sys Writes:      0 Other Phy Rd/Wt: 0/ 0
VoluntaryCSwitch     : 4 Raw Reads :      0 Proc Start Time
Running CPU          : 0 Raw Writes:      0 Tue Mar 16 15:49:14 2004
CPU Switches         : 0 Bytes Xfer: 0kb
  
```

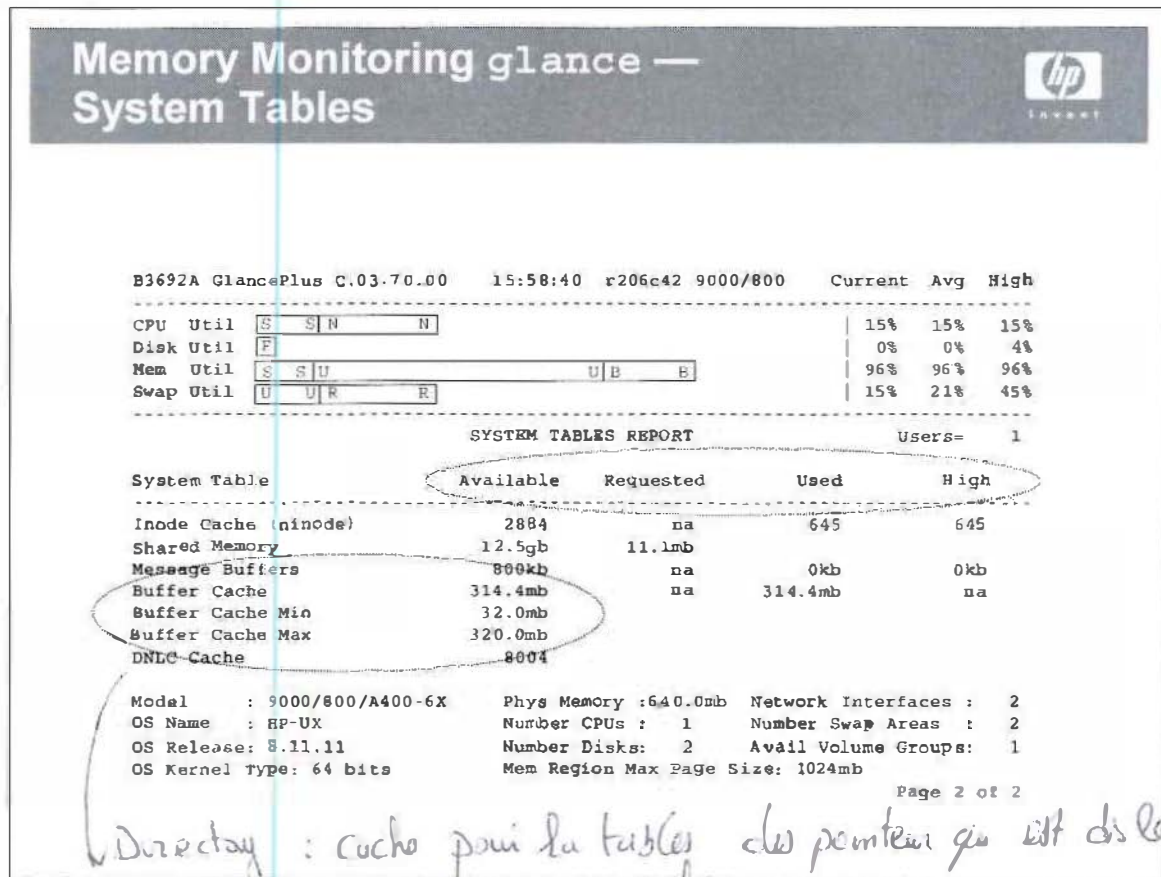
C - cum/interval toggle % - pct/absolute toggle
Page 1 of 1

Student Notes

The **glance** Individual Process report displays memory usage for an individual process, and the RSS and VSS sizes for the process.

Also displayed on a per-process basis, is the VM reads and VM writes being performed by the process. This indicates how much paging from/to the swap device the individual process is performing. If performance is poor for an individual process, this is a good field to check.

6-13. SLIDE: Memory Monitoring glance — System Tables



Student Notes

The **glance** System Table report displays the size of kernel tables in memory, and the current utilization of these tables. It is important not to set the size of these tables too large, as the tables are memory resident (and the bigger the table, the more memory it consumes). Yet, it is even more important that enough resources be allocated so that the kernel does not have to wait for a resource to become free (or even error out) when a particular resource is requested.

The Available column displays the total size of the particular table, and the Used column shows how many entries within the table are currently being used. In general, the Used value should not be close to the Available value. If it is, then the kernel is close to running out of that particular resource.

The High % column shows the high water mark for the resource since **glance** has been running.

Also of interest in this report are the buffer cache statistics, especially the Buffer Cache that shows the current size of the buffer cache.

Module 6
Memory Management

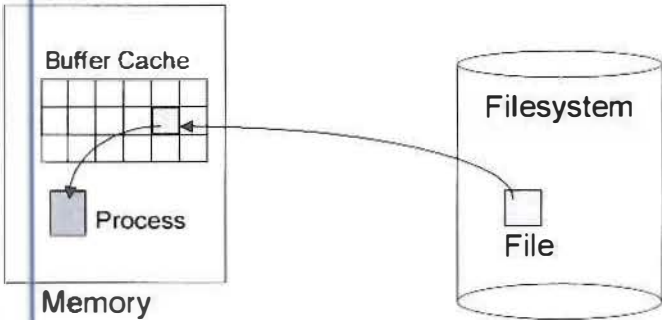
NOTE: There are two pages to this report. Shown here is the second page of this report. More system tables are shown on the first page.

6-4. SLIDE: The Buffer Cache

The Buffer Cache



- Pool of memory designed to retain the most commonly accessed files from disk
- Used only for file system I/O (not raw I/O)
- Size of buffer cache controlled by `dbc_min_pct` and `dbc_max_pct`



Student Notes

The buffer cache exists to speed up file system I/O. The system tries to minimize disk access by going to disk as infrequently as possible, because disk access is often a bottleneck on most systems. Therefore, the most recently- or commonly-accessed files from disk persist in the portion of memory called the **buffer cache**.

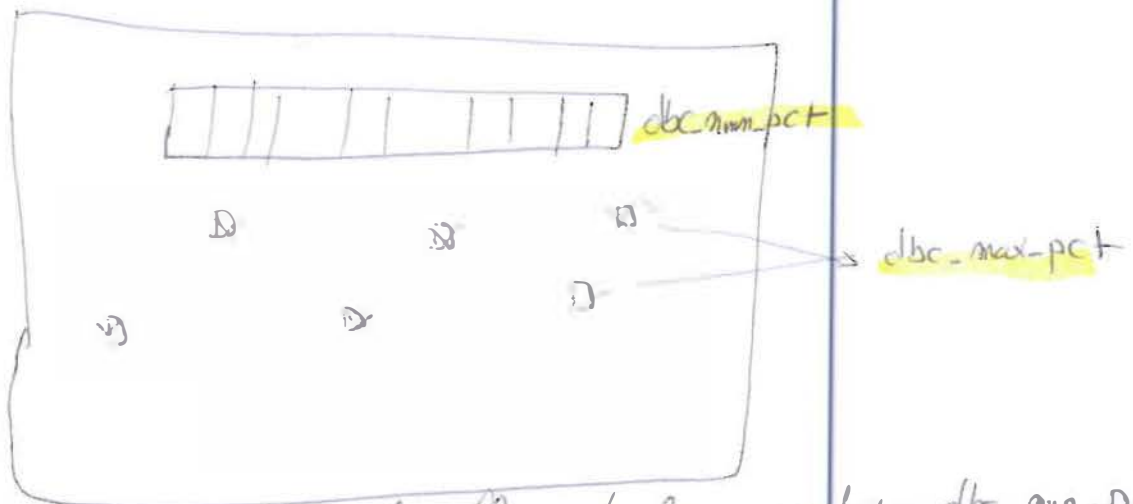
It is called dynamic because the size of the buffer cache grows or shrinks dynamically, depending on competing requests for system memory. Its minimum size is governed by the tunable parameter `dbc_min_pct`, and it cannot grow larger than the size specified in `dbc_max_pct`. These two parameters are expressed as percentages of total physical memory on the system.

Let's say `dbc_min_pct` is set to 10, while `dbc_max_pct` is 50. This means that initially 10% of physical memory is allocated to the buffer cache. As the system needs more space to buffer files read in from disk, the buffer cache will allocate more memory, and this will continue until it occupies 50% of memory, its maximum size. Later, when the system requires more memory for another use, say processes, the buffer cache could shrink an appropriate amount, but will never be less than the 10% minimum value. Therefore, a larger buffer cache is able to hold more files and will minimize their access time but will leave less memory available for other uses.

Module 6
Memory Management

NOTE: The buffer cache is dynamic in nature only when two other tunable parameters, **bufpages** and **nbuf**, are both set to their default values of 0.

Another example: if **dbc_min_pct** and **dbc_max_pct** are both set to the same value, say 20, the kernel will always use exactly that percentage of physical memory for the buffer cache.

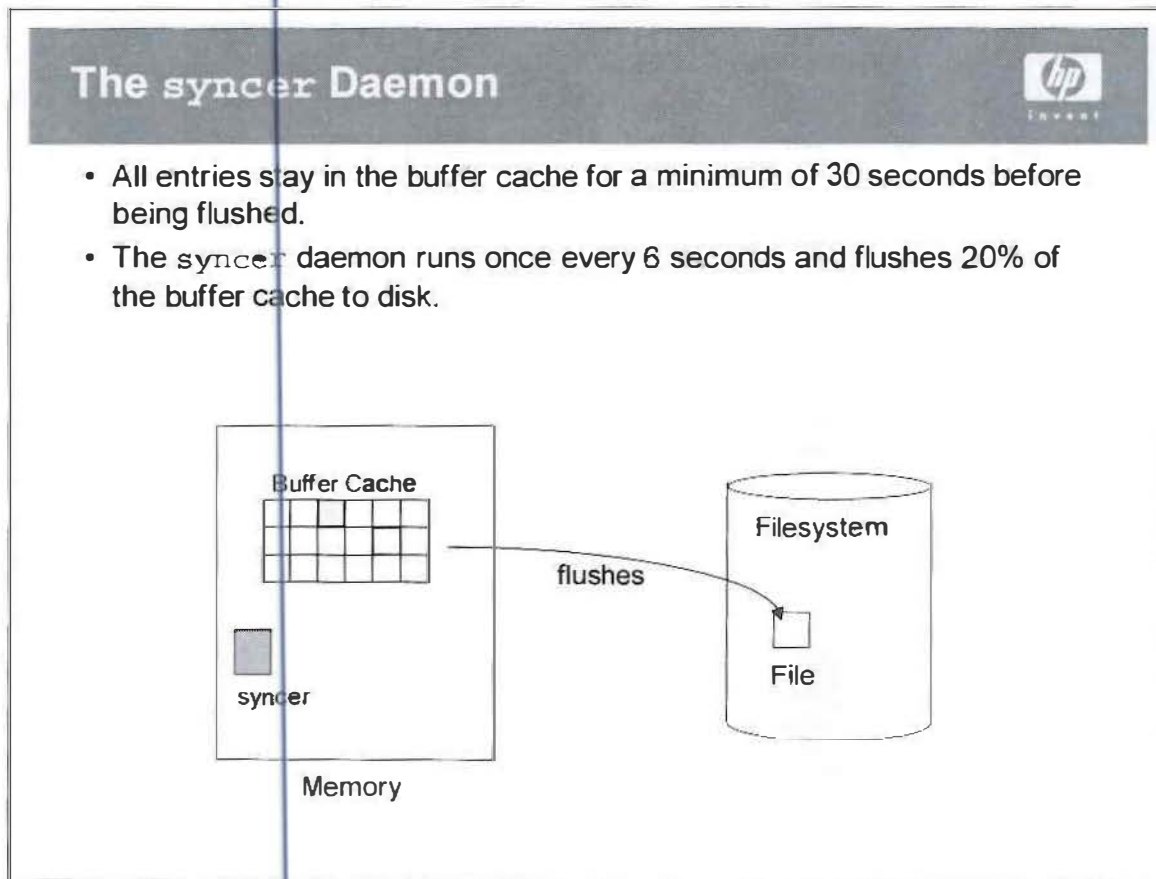


en fait de la même le buffer cache alloue de façon continue **dbc_min_pct**

deux autres paramètres $\left. \begin{array}{l} \frac{nbuf}{bufpage} \end{array} \right\} =$ priorité de kernel

mmode n'est pas utilisée maintenant il y'a le vxmmode

6-5. SLIDE: The syncer Daemon



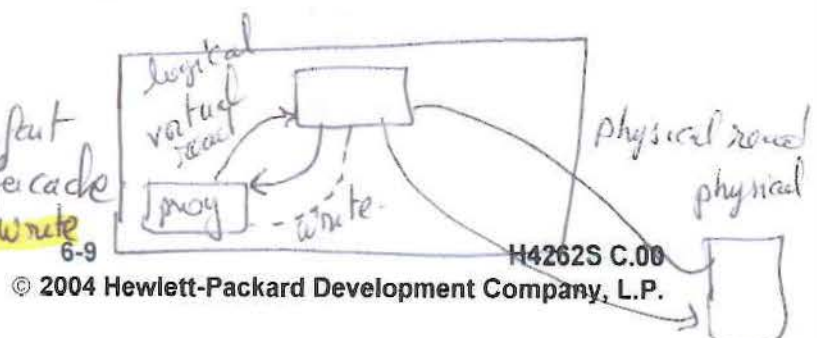
Student Notes

For disk writes, data flows from the buffer cache to disk. How does it get to the buffer cache? The kernel writes data to it. The **syncer** process takes care of flushing data in the buffer cache to the files on the disk.

When a user edits a file, makes changes to that file, and saves the changes, those changes do not go to disk right away. The kernel writes the data to the buffer cache, and some time later (within 60 seconds) the data finally arrives at the disk. This time period is chosen as a balance between ensuring that the file system is fairly up-to-date in case of a crash and efficiently performing disk I/O.

There are many applications that do not rely on the operating system's built-in processes to flush data to disk, but instead take over that operation themselves. In other words, they create their own buffers and manage the flushing at appropriate intervals. A common example is a database application that needs to guarantee the completion of a transaction within a specified time interval.

Si write de prog dans buffer cache
Si espace pas dans le buffer cache fait
un sync pour vide le buffer cache
il sera considéré comme **write**
<http://education.hp.com>
Si espace existe il sera considéré
comme logical write

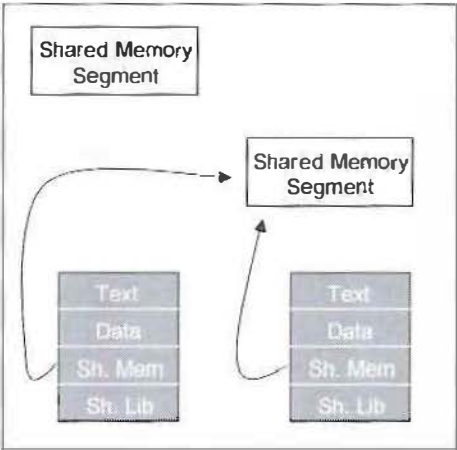


6-6. SLIDE: IPC Memory Allocation

IPC Memory Allocation



```
# ipcs -mob
IPC status from /dev/kmem as of Sat Feb 14 06:53:27 1998
T      ID      KEY      MODE      OWNER      GROUP      NATTCH      SEGSZ
Shared Memory:
m      5 0x06347849 --rw-rw-rw-   root      root       0           77384
m      7 0x000c0568 --rw-----   root      root       2          131516
```



Memory

Student Notes

UNIX implements interprocess communications using different mechanisms. Three mechanisms that require additional system memory are semaphores, shared memory, and message queues.

- **Semaphores** are used to synchronize memory resources between competing processes.
- **Shared memory** segments are resources capable of holding (in memory) large amounts of data that can be shared between processes.
- **Message queues** hold strings of information (messages) that can be transferred between processes. Two types of processes that utilize message queues are networking and database processes.

Shared memory provides a mechanism to reduce interprocess communication costs significantly. Two processes that are ready to share data, address the same portion of shared memory into their addressable space. Changes made to the shared memory are seen immediately by all processes and do not require kernel services. So from a kernel perspective, other than initially setting up the shared memory, there is very low cost in using shared memory.

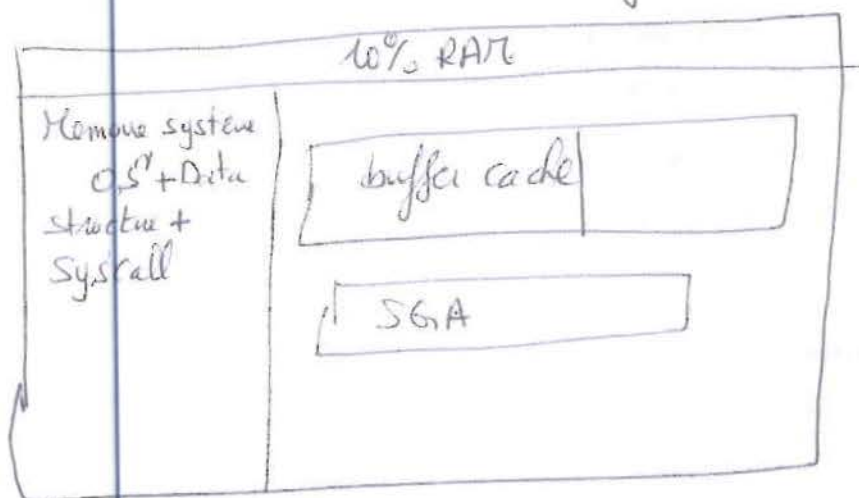
On the slide, each process has a shared memory segment that references one and the same shared memory area.

The more processes that allocate shared memory segments, the higher the memory usage. The shared memory segments in physical memory can be viewed with the `ipcs -mob` command or a reporting tool like `glance`. From time-to-time, they might have to be cleaned up or removed manually if an application terminates ungracefully. This is done by the *superuser* with the `ipcrm` command.

A worthwhile baseline measurement for a system administrator is to run the `ipcs -mob` command during a quiet period. It is also eye opening to repeat this command when the system is at its busiest.

`ipcrm 0x06347849`

Exemple : je veux allouer 500 Mo pour SGA d'oracle
pour 2 GB



1/4 RAM.

pour 266 B de RAM.
c'est pas fixe par exemple.

Res aux = 200 Mo
system = 500 Mo
buffer cache = 50% 1024 Mo
SGA = 200 Mo
user = 100 Mo

Lockable memory = 10% du mémoire + 1/4 RAM
grep -i physical /var/adm/syslog

6-7. SLIDE: Memory Metrics to Monitor — Systemwide

Memory Metrics to Monitor — Systemwide



- Is vhand Active? → scan & liberer plus pages et active
 - Pages scanned by vhand (SR)
 - Pages freed by vhand (FR)
 - Pages paged out
- Is swapper Active?
 - Processes deactivated (SO)
 - Amount of free memory relative to
 - lotsfree
 - desfree
 - minfree
- Size of dynamic buffer cache
- Size of IPC Shared memory segments

si j'ai des desactivable mémoire donc j'ai des pages gelées
mémoire

Student Notes

The utilization of memory can be monitored in a number of different ways. There are multiple tools and multiple metrics that monitor memory usage. The first metrics you want to look at are those that will tell you whether **vhand** is active.

Pages Scanned by vhand

This is the number of pages the **vhand** process has scanned (i.e. dereferenced with the reference hand) when looking for pages to free in memory. This tells you that **vhand** is actively scanning pages in an attempt to free them up. There is some memory pressure.

Pages Freed by vhand

This is the number of pages the **vhand** process has freed (i.e. the reference bit was still dereferenced when the free hand looked at it). The ratio between *pages scanned* and *pages freed* indicates how successful the **vhand** process is when looking for memory pages to free.

Amount of Paging

This indicates the level of disk activity to the swap partition. If a consistent amount of paging to swap space is occurring, then performance is impacted (most likely significantly).

Next, check to see if the `swapper` is active.

Process Deactivations

This indicates that processes are being deactivated, meaning free memory has fallen below the MINFREE threshold. There is severe memory pressure.

Amount of Free Memory

This indicates the severity of the free memory situation. If free memory has fallen below LOTSFREE, then we know some paging has taken place. `vhand` is active. If it is below DESFREE, then the situation is more severe, and much more paging is occurring. `vhand` is aggressively active. Finally, if free memory is below MINFREE, then a high level of paging *and* process deactivation is occurring. `vhand` and `swapper` are both active. To determine what the values are for lotsfree, desfree, and minfree, use the following commands:

```
# echo "lotsfree/D" | adb -k /stand/vmunix /dev/mem
# echo "desfree/D" | adb -k /stand/vmunix /dev/mem
# echo "minfree/D" | adb -k /stand/vmunix /dev/mem
```

The settings for these three values in the kernel will then be displayed in 4K pages. You can then compare them to the current size of the free page list. These values will not change, unless you change the size of Non-Kernel Memory. (Remember the formulas shown earlier?)

Size of Dynamic Buffer Cache

This is the amount of memory being consumed by the buffer cache. If memory is full and the buffer cache is large, it will most likely cause paging, since the buffer cache typically shrinks slower than the rate at which new memory is needed. Heavy disk I/O demands may prevent the buffer cache from shrinking at all.

Size of IPC Shared Memory Segments

This is the amount of memory used for interprocess communications. Of special interest will be the number and sizes of shared memory segments, as these can be quite large, especially if graphical applications or a database management system is being used.

6-8. SLIDE: Memory Metrics to Monitor — per Process

Memory Metrics to Monitor — per Process



- Size of RSS/VSS
- Size of text, data, and stack segments
- Number of shared memory segments
- Amount of time blocked on virtual memory

Student Notes

Individual processes vary greatly in terms of the amount of memory they use. Metrics to monitor memory utilization on a per-process basis include the following:

Size of RSS/VSS

The Resident Set Size (RSS) for a process is the portion of the process (in KB) that is currently resident in physical memory. Since the entire process does not have to be resident in memory in order to execute, this shows how much of the process is actually resident in memory.

The Virtual Set Size (VSS) for a process is the total size of the process (in KB). This indicates that if the entire process were to be loaded, this is how much memory the entire process would consume. Very rarely is the entire process resident in memory. If the entire process were in memory, then the RSS value would be equal to the VSS value.

Size of Text, Data, and Stack Segments

These are the RSS and VSS sizes for the three main components of a process. Since every process has a single text, data, and stack segment, these values should be monitored, especially for large processes. The data segment is the most likely to be large.

Each of these three segments has a maximum size to which they can grow – limited by tunable kernel parameters. They are `maxtsiz`, `maxdsiz`, and `maxssiz` for a 32-bit process. They are `maxtsiz_64bit`, `maxdsiz_64bit`, and `maxssiz_64bit` for a 64-bit process. If a process tries to grow one of these segments beyond its maximum size, then the process terminates (and in some cases “core dumps”).

Number and Size of Shared Memory Segments

These are the shared memory segments to which a process is attached. The maximum size of a shared memory segment is limited by the kernel parameter, `shmmax`. The number of shared memory segments a process can attach to is limited by the kernel parameter, `shmseg`.

Amount of Time Spent Blocked on Virtual Memory

This is the amount of time the process was prevented from executing because it was waiting (or blocked) on a text or data page to be paged in.

6-9. SLIDE: Memory Monitoring vmstat Output

Memory Monitoring vmstat Output

```
#=> vmstat -n 5
```

VM

memory						page					faults	
avm	free	re	at	pi	po	fr	de	sr	in	sy	cs	
9140	3824	3	4	0	0	0	0	0	675	824	140	

CPU

cpu			procs														
us	sy	id	r	b	w												
9	5	86	1	100	0												
9017	3500	41	49	11	0	0	0	0	1257	2823	329						
24	17	60	0	100	0	41	0	0	0	1419	3795	481					
10292	2255	65	20	41	0	0	0	0	1419	3795	481						
67	24	9	5	102	0	10227	976	89	19	85	0	0	0	1698	4771	641	
67	33	0	7	103	0	10958	400	81	12	91	48	26	0	194	1791	5847	697
67	31	3	8	110	0	10759	454	33	3	98	51	24	0	268	1598	4313	598
62	20	18	6	111	0	13448	404	21	0	65	74	39	0	282	1021	3175	354
32	15	53	0	118	0												

Student Notes

A useful command to view virtual memory statistics is **vmstat**. The slide shows **vmstat**'s output being updated every 5 seconds.

When viewing **vmstat**'s output, always keep an eye on the **po** (pages paged out) parameter. Ideally, you want this to be zero, indicating no paging out is occurring.

Statistics regarding the **vhand** algorithm, the **fr** (pages freed per second) and **sr** (pages scanned by the clock algorithm, per second) parameters show the actual behavior of **vhand**.

Output Headings

procs

- r** In run queue
- b** Blocked for resources (I/O, paging, and so on)
- w** Runnable or short sleeper (less than 20 seconds) but deactivated

memory

avm	Active virtual pages (run during the last 20 seconds)
free	Size of free list page (in 4K pages)
re	Page reclaims per second
at	Address translation faults per second (page faults)
pi	Pages paged in per second
po	Pages paged out per second
fr	Pages freed per second
defacto	Anticipated short term memory shortfall
sr	Pages scanned by algorithm per second

faults

in	Non-clock device interrupts per second
sy	System calls per second
cs	CPU context switches per second

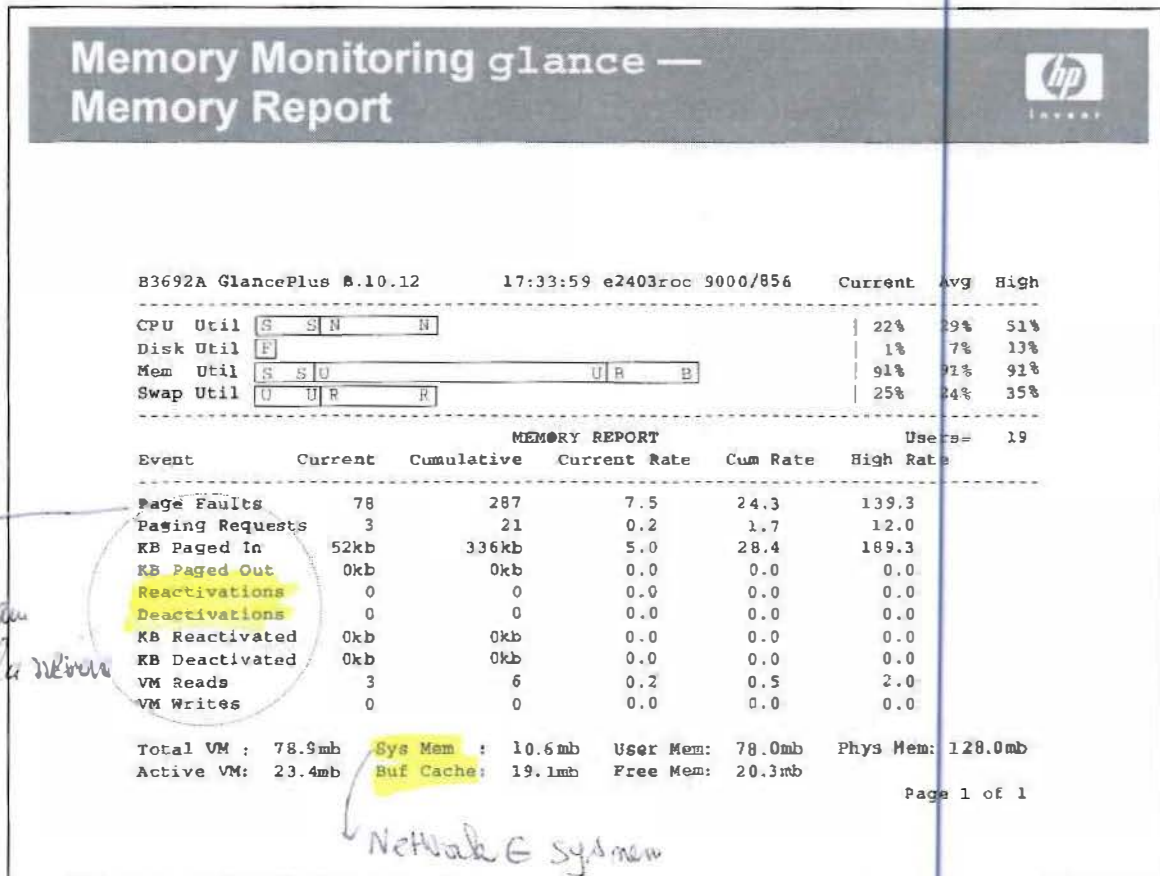
CPU

us	Percentage of time CPU spent in user mode
sy	Percentage of time CPU spent in system mode
id	Percentage of time CPU is idle

with -S option

si	Processes reactivated per second
so	Processes deactivated per second

6-10. SLIDE: Memory Monitoring glance — Memory Report

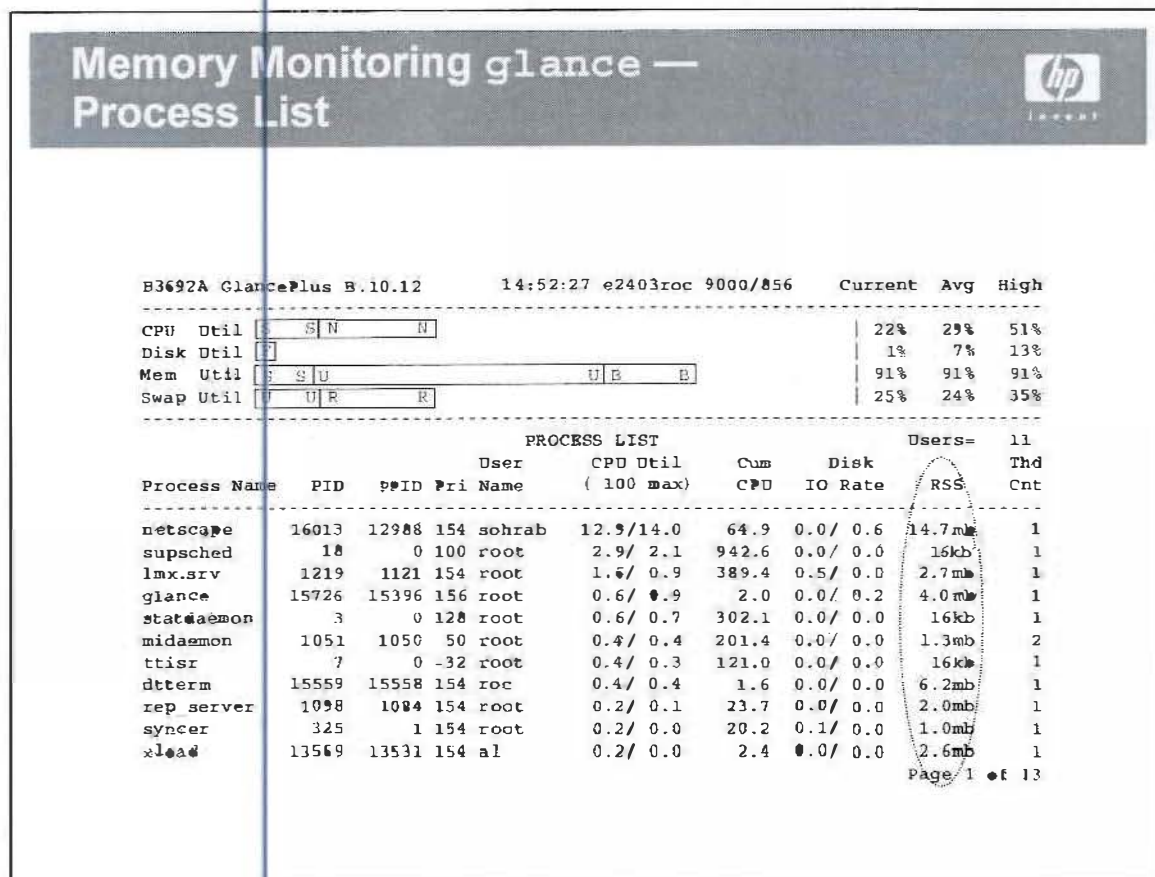


Student Notes

glance has extensive memory monitoring abilities. Like `vmstat`, it can give paging statistics, in addition to showing if any processes are being deactivated. Remember, this is an indication of severe memory shortage.

There is other valuable information on this report, such as the statistics at the bottom showing the current Dynamic Buffer Cache size, the current amount of Free Memory, and the total Physical Memory in the system.

6-11. SLIDE: Memory Monitoring glance — Process List



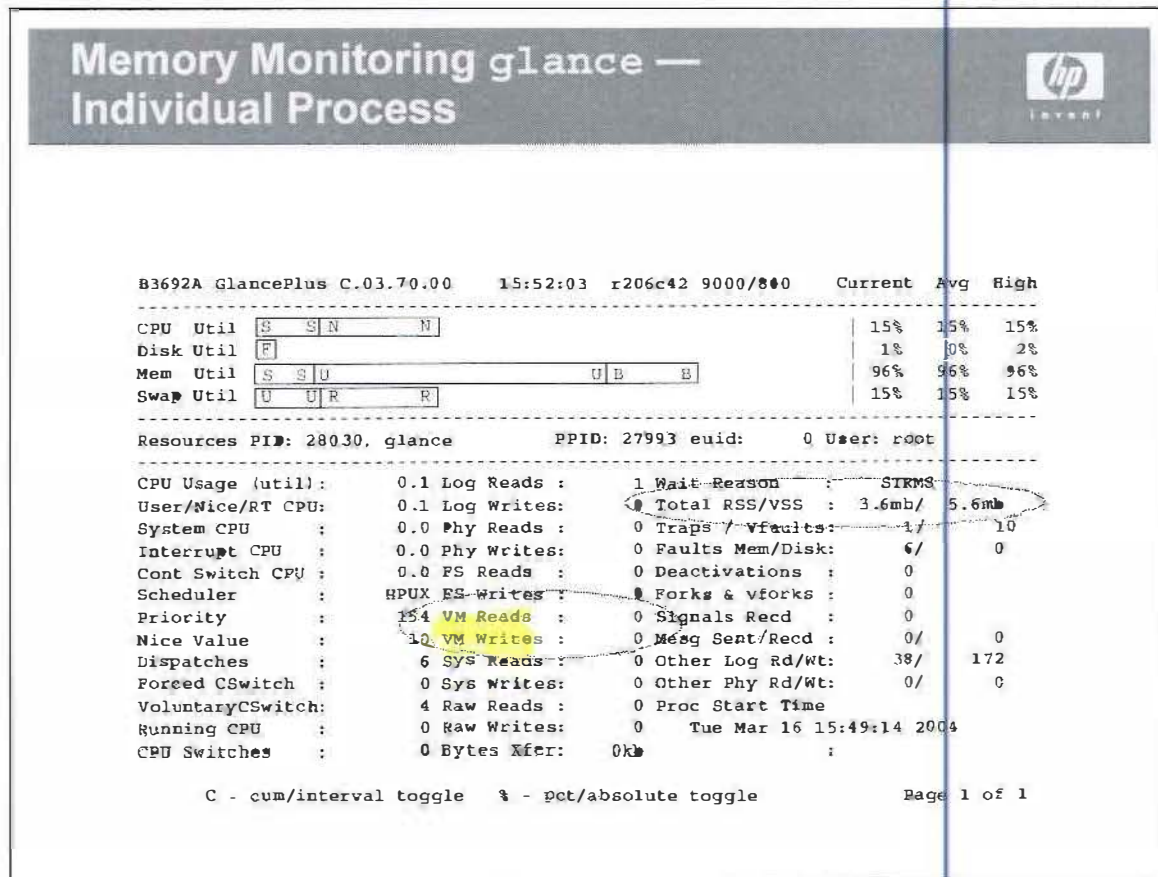
Student Notes

The **glance Process List** report can be used to monitor process statistics, including how much memory processes are currently consuming. The highlighted column, **RSS (Resident Set Size)**, shows memory being used on a per-process basis. Very simply put, this helps to identify the "memory hogs" on the system.

For example, the process called **netscape** has an RSS of 14.7 MB, while **statd** is minimal. Other large processes include **glance**, **xload**, and **dtterm**. What do all these processes have in common? They are all GUI (graphical user interface) programs running as windows in a graphical window environment.

Moral: programs that open their own windows are relatively memory-intensive and should be minimized. Users should be encouraged not to leave several windows open on their screens if they do not have a continuing need for them.

6-12. SLIDE: Memory Monitoring glance — Individual Process

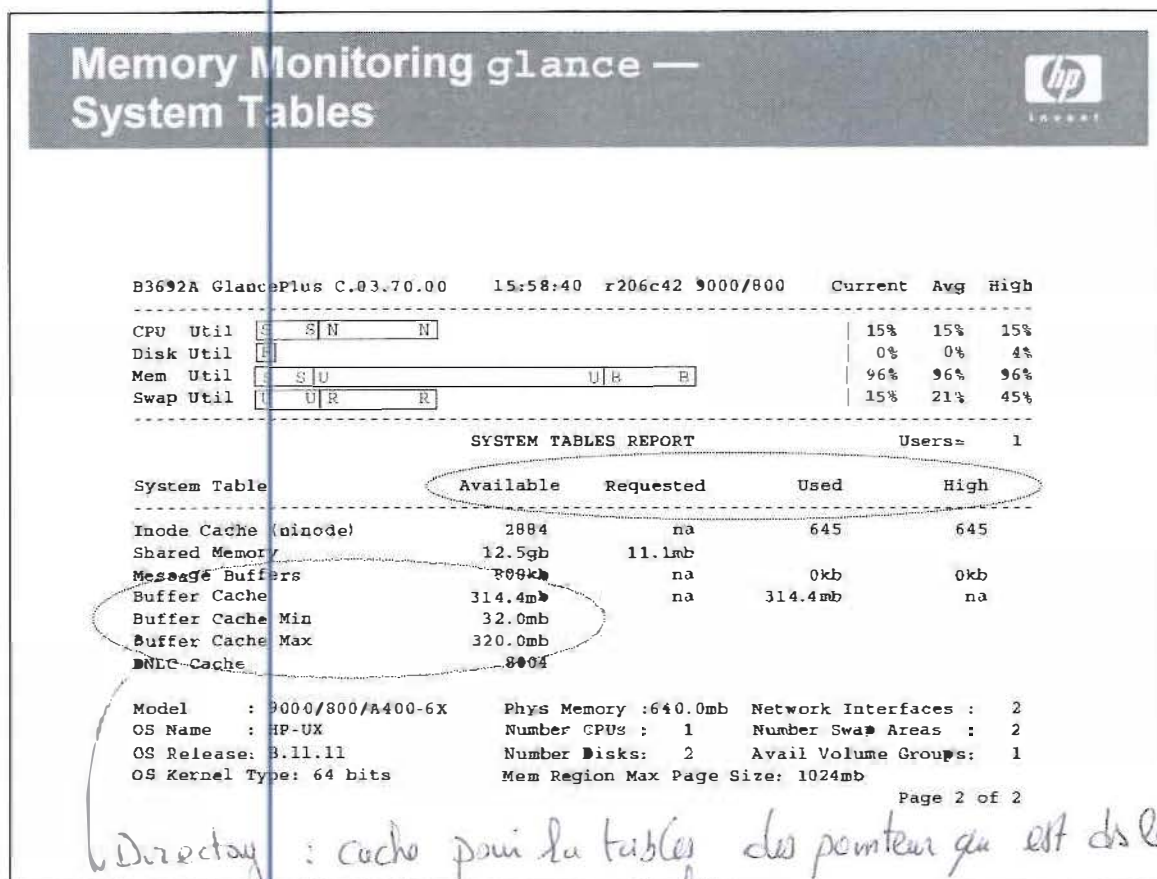


Student Notes

The **glance** Individual Process report displays memory usage for an individual process, and the RSS and VSS sizes for the process.

Also displayed on a per-process basis, is the VM reads and VM writes being performed by the process. This indicates how much paging from/to the swap device the individual process is performing. If performance is poor for an individual process, this is a good field to check.

6-13. SLIDE: Memory Monitoring glance — System Tables



Student Notes

The **glance** System Table report displays the size of kernel tables in memory, and the current utilization of these tables. It is important not to set the size of these tables too large, as the tables are memory resident (and the bigger the table, the more memory it consumes). Yet, it is even more important that enough resources be allocated so that the kernel does not have to wait for a resource to become free (or even error out) when a particular resource is requested.

The Available column displays the total size of the particular table, and the Used column shows how many entries within the table are currently being used. In general, the Used value should not be close to the Available value. If it is, then the kernel is close to running out of that particular resource.

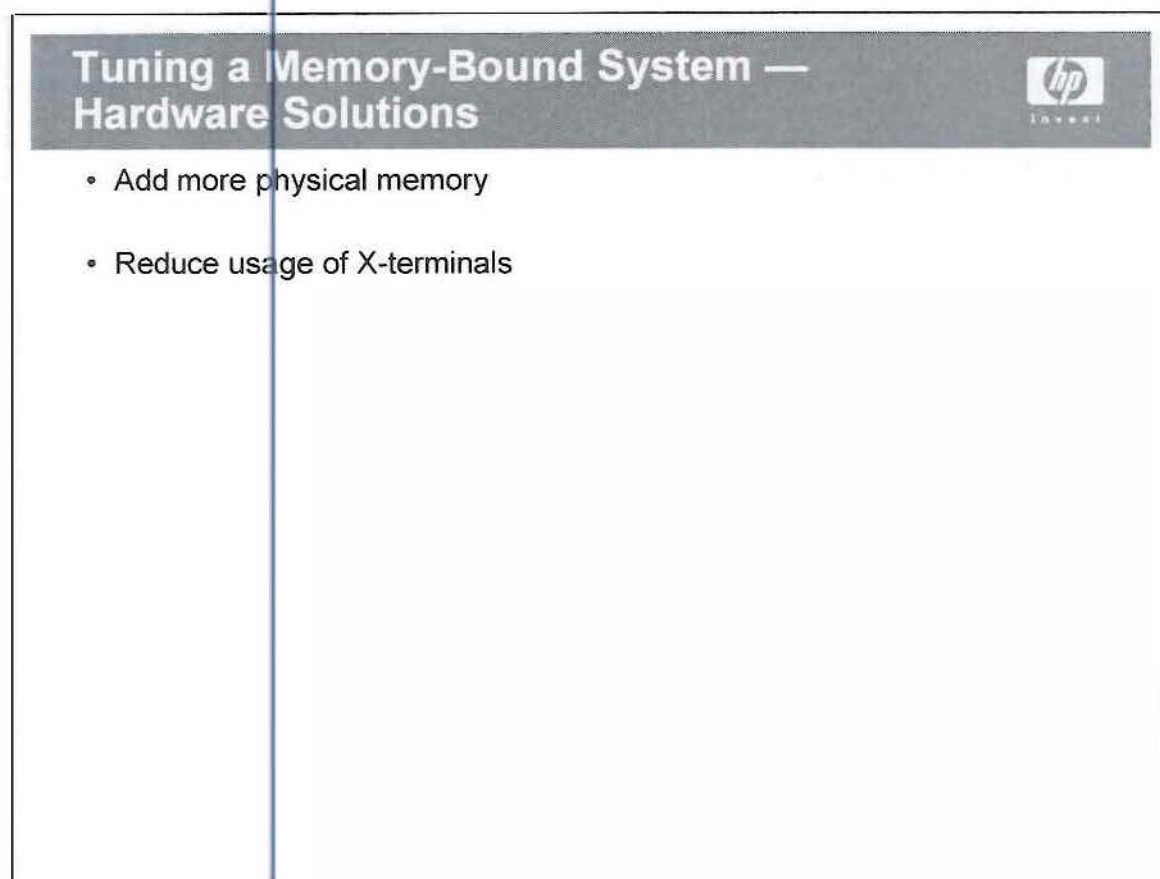
The High % column shows the high water mark for the resource since **glance** has been running.

Also of interest in this report are the buffer cache statistics, especially the Buffer Cache that shows the current size of the buffer cache.

Module 6
Memory Management

NOTE: There are two pages to this report. Shown here is the second page of this report. More system tables are shown on the first page.

6-14. SLIDE: Tuning a Memory-Bound System — Hardware Solutions



The slide features a dark grey header with the title "Tuning a Memory-Bound System — Hardware Solutions" in white text. To the right of the title is the HP logo with the word "Invest" underneath it. Below the header, on a white background, are two bullet points:

- Add more physical memory
- Reduce usage of X-terminals

Student Notes

An obvious hardware solution to a memory bottleneck is to add more physical memory. While this solution requires an outlay of money, it may pay for itself quickly by saving the system administrator hours of time looking for ways to reduce memory consumption.

If adding more memory is not an option, then a second hardware suggestion is to look at the use of X terminals on the system. An X terminal typically consumes a large portion of memory. X terminals will take up 34 MB of memory for *light* application usage, and as much as 1020+ MB for *heavy* application usage.

These figures do *not* take into account any additional RAM that the system will use for window managers or any other X-related overhead.

6-15. SLIDE: Tuning a Memory-Bound System — Software Solutions

Tuning a Memory-Bound System — Software Solutions



- Look for unnecessary processes
 - Extra windows
 - Screen Savers
 - Long strings of child processes
- Reduce `dbc_max_pct` (max size of dynamic buffer cache).
- Identify programs with memory leaks.
- Check for unreferenced shared memory segments.
- Use `serialize` command to reduce process thrashing.
- Use PRM to prioritize memory allocation.

Student Notes

Quite often, users will run X-windows type programs to enhance the look of their desktop. Examples include an X-eyes program, a bouncing ball program, or fancy screen savers. All of these graphical programs consume system resources, including memory.

The biggest consumer of memory will most likely be the buffer cache. We saw earlier that if the buffer cache is dynamic, it will grow to its maximum size, as long as memory is available. The problem with this is when a process needs additional memory, and the free memory is below `LOTSFREE`, then the buffer cache is slow to shrink (if at all!), causing paging to occur among the processes. To prevent this situation, the tunable parameter `dbc_max_pct` should be tuned to limit the maximum size in which the buffer cache can grow. A recommendation for `dbc_max_pct` is 25 or less.

Programs with memory leaks will allocate memory and then stop using – without returning it to the system for use elsewhere. These programs may require you to shut them down periodically, to release the memory. They may even require you to reboot the system occasionally to reclaim the memory. There are a number of third party tools that will help you locate memory leaks in applications – such as Purify.

Unreferenced shared memory segments can also be a problem. An application sets one up and then forgets to deallocate it when the application exits. Here is a possible procedure for locating abandoned shared memory segments:

First, look for any shared memory segments that have no processes attached to them.

```
# ipcs -ma
```

Note which shared memory segments have a "0" in the NATTCH column. If they are owned by "root", let them stay. Otherwise, write down their ID numbers and their CPID numbers.

Second, one at a time, find out whether the creating process still exists.

```
# ps -el | grep <CPID number>
```

If it does, it's probably just a quiescent segment. But if not, the segment is probably abandoned.

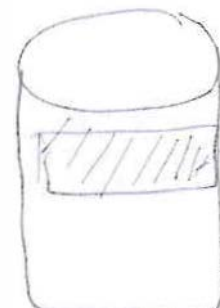
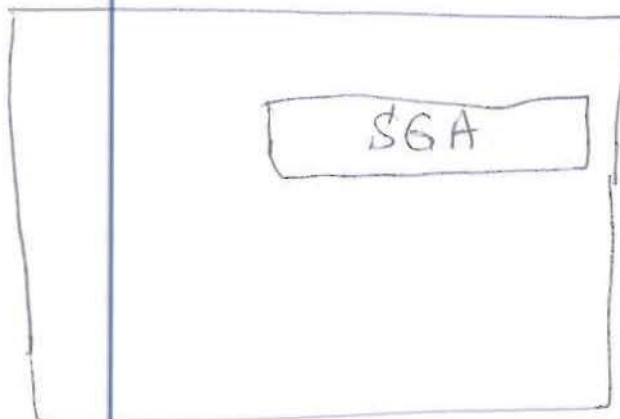
Finally, remove the segment.

```
# ipcrm -m <ID number>
```

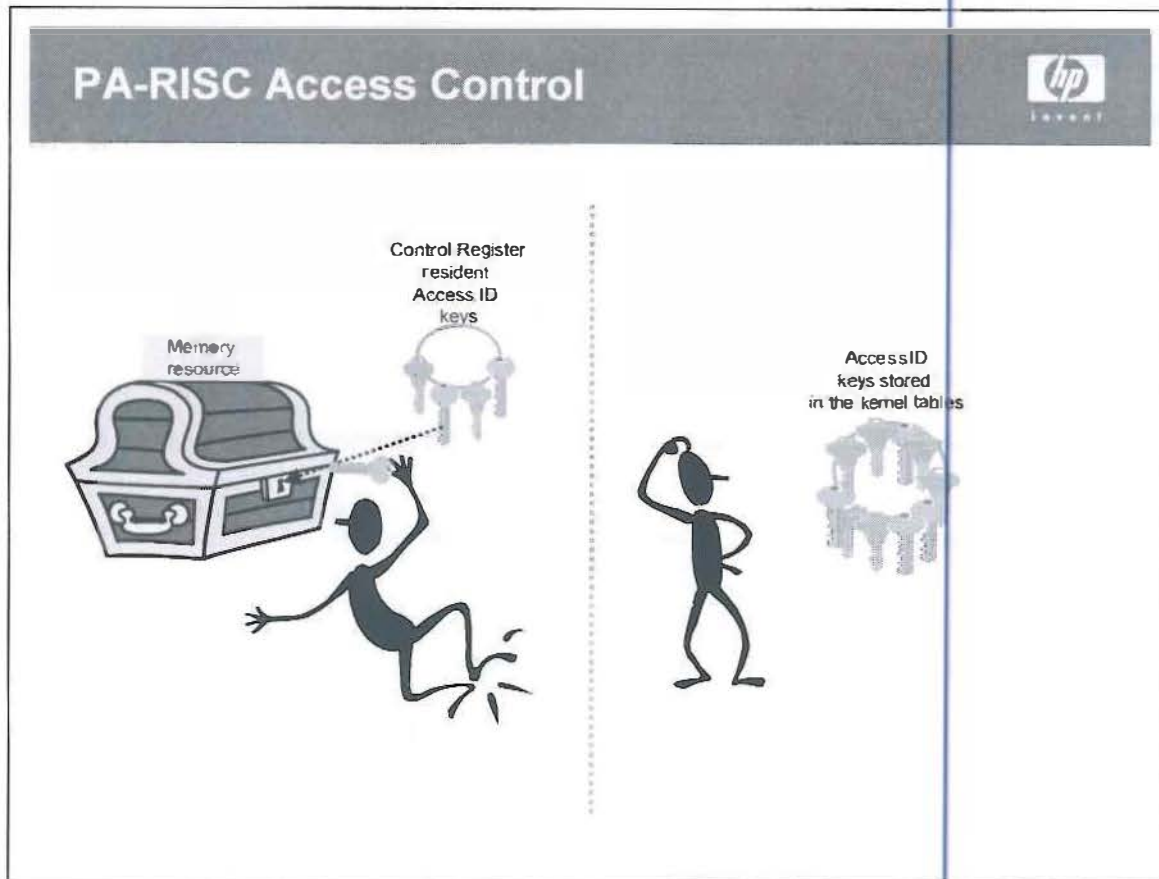
pow m
pow s
ipcrm -s

The serialize command will be discussed later in this chapter.

You may wish to use PRM to control your memory resource and its allocation.



6-16: SLIDE: PA-RISC Access Control



Student Notes

Since we are discussing system memory and performance there is one other topic that we should think about, hardware based memory page access control.

The processor architecture has several features related to assuring that a process thread can not access areas of physical memory that are not part of its process space. An in depth discussion of page access control is presented in the HP-UX training course, "Inside HP-UX", course number H5081S and we won't attempt to recreate it here.

There is one particular aspect of this hardware feature that we will spend some time with in discussion though, and that is "Protection ID's".

Every discrete region of virtual memory assigned to a process (text space, private data space, shared memory space, shared library data space, etc) is assigned a unique ID "key", called an Access Key. Any process attempting to access that memory space must have a copy of a matching ID "key", called a Protection Key. To speed things up, the most frequently or likely used Protection Keys are kept in processor registers. (These registers are part of a process thread's "context" and are preserved across switches and interrupts.) The hardware performs the Protection check as part of the actual memory access instruction.

Now here is the catch, there is only room in the control registers for a limited number of frequently used Protection Keys. The rest are stored in kernel space in memory management tables, which are accessed when a protection ID fault occurs. The fault handler will search for and find these other "keys" when they are needed but at the cost of CPU cycles!

To better understand the dynamics of this process consider the following analogy:

The Key Ring

I have many keys to many locks around my home and office. It is not practical to carry all of my keys around with me all the time due to their bulk and weight. To solve this problem I have two key rings. One is small and has only those keys that I need on a daily basis, my car key, house key, desk key, and garage key. The other key ring is large and bulky with dozens of other miscellaneous keys; my workshop, tool boxes, garden shed, lawnmower (wish I could loose that one!), boat ignition, etc...

This method is a blessing and a curse. When I need to start the car or unlock the front door the key I need is readily available in my pocket and I can quickly gain access. When I actually have time to go fishing, it is always a hassle to go find my utility key ring and remember to take the boat key with me. (Once I actually hauled the boat all the way to the lake, several miles away from my home, only to realize that I had not remembered the boat key!).

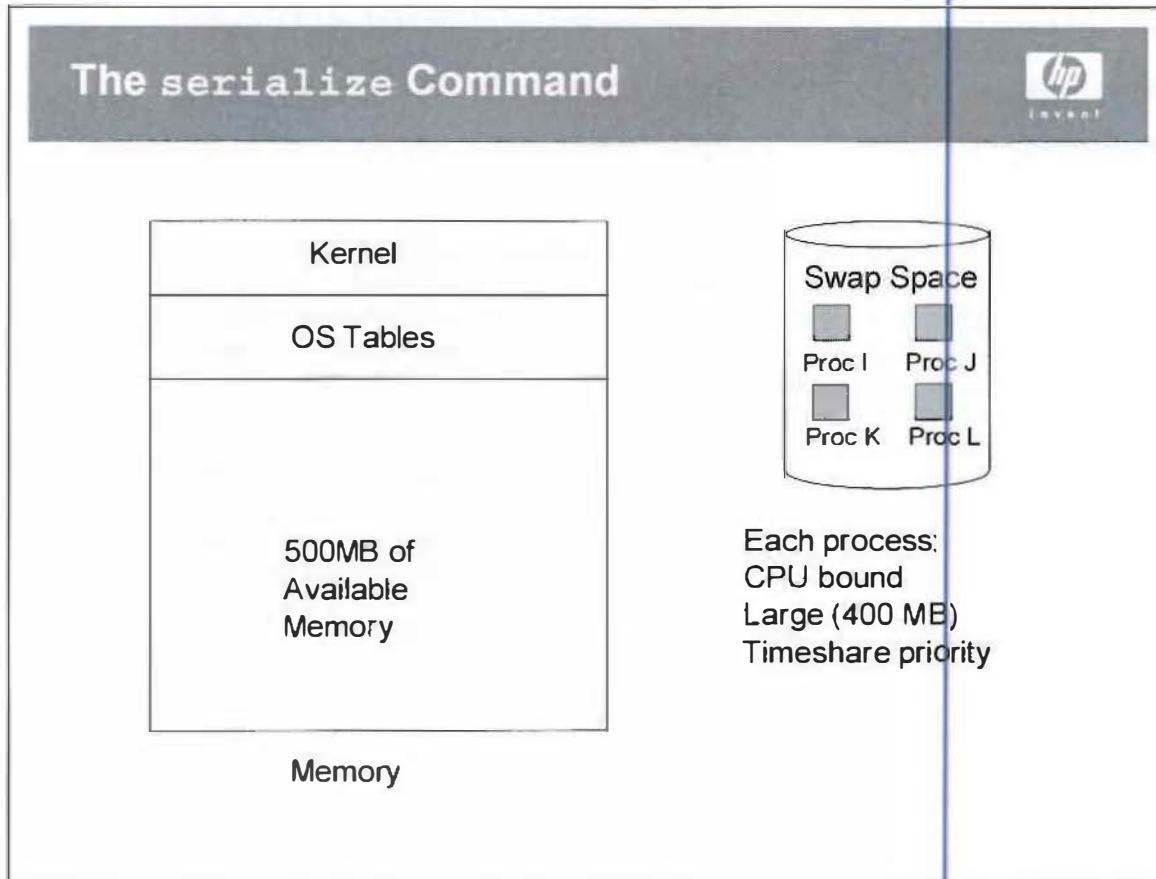
To somewhat address this problem I now move the boat key to my everyday key ring during the summer months (replacing the snow-blower key) and reverse the procedure in the fall.

The HP-UX kernel performs a similar process, every time a protection ID fault occurs. The fault handler moves the key it had to search for to the register context of the thread (replacing the least recently used key).

PA-RISC 1.x has room for 4 keys in the register context while PA-RISC 2.X has room for 8 keys. IA-64 has room for at least 16 keys. Depending on how frequently a process moves from one memory region to another the number of protection ID faults will vary. With the larger number of Protection Registers in the later processors, Protection Register thrashing has become much less a problem than it has been in the past.

It should also be noted that shared library regions on 11.x were modified to use a type of "skeleton" key, i.e., a key that always matches so that attempted access to them will never result in a protection ID fault.

6-17. SLIDE: The serialize Command



Student Notes

The **serialize** (1) command can help if a system has a number of large processes and is experiencing memory pressures. The **serialize** command will allow these big processes to run one after another, instead of running all at the same time. By running the processes sequentially, rather than in parallel, the CPU can spend more time executing the process code (i.e. user mode) and less time managing the competing processes (i.e. kernel mode).

Thrashing

On systems with very demanding memory needs (for example, systems that run many large processes), the paging daemons can become so busy moving pages in and out that the system spends too much time paging and not enough time running processes.

When this happens, system performance degrades rapidly, sometimes to such a degree that nothing seems to be happening. At this point, the system is said to be **thrashing**, meaning it is doing more overhead work than productive work.

How `serialize` Helps Reduce Thrashing

All processes marked via the `serialize` command will run serially with other processes marked the same way. The `serialize` command addresses the problem caused when a group of large processes all try to make forward progress at once, which results in degrading throughput. In such a case, each process constantly faults in its working set, only to have the pages stolen when another process starts running. By using the `serialize` command to run large processes one at a time, the system can make more efficient use of the CPU, as well as system memory.

Let's look at the example on the slide. We have a system with 500MB of available memory. We are trying to execute four processes. Each process is CPU bound, has large memory requirements (400MB), and has a timeshare priority level.

The first process (**I**) executes. As it executes, its pages are faulted into memory. At the end of its timeslice (typically 100ms), it is switched out and process **J** is started. As it executes, it pages in a large number of pages – forcing the pages belonging to process **I** to be paged out. 100ms later process **J** is switched out and process **K** starts up, pulling its pages into memory and pushing the other process's pages out.

The system spends so much time pulling pages in and pushing pages out, that it literally has no time left to perform any useful work. The culprit here is the timeslice. OK, we could simply disable timeslicing altogether via the tunable parameter (`timeslice`). But that may be overkill – more than we want to do. After all, it's just these four processes that are causing the thrashing. A better solution would be to "serialize" these processes. When you do that, each process executes until it either voluntarily gives up the CPU or it is preempted by a stronger priority process – which will happen much less frequently than the timeslice! Thus more real work will get done and much less paging will be needed.

In 10.20, the kernel was given the authority to serialize processes automatically, if it detects that memory thrashing is taking place and it can identify which processes are responsible for the thrashing.

6-18. LAB: Memory Leaks

There are several performance issues related to memory management, memory leaks, and swapping/paging, protection ID thrashing.... Let's investigate a few of them.

1. Change directories to `/home/h4262/memory/leak`:

```
# cd /home/h4262/memory/leak
```

Memory leaks occur when a process requests memory (typically through the `malloc()` or `shmget()` calls) but doesn't free the memory once it finishes using it. The five processes in this directory all have memory leaks to different degrees.

2. Before starting the background processes, look up the current value for `maxdsiz` using the `kmtune` command on `11i v1` and the `ktune` command on `11i v2`.

On the `rp2430`:

40000000

```
# kmtune -lq maxdsiz
```

On the `rx2600`:

```
# ktune -avq maxdsiz (by default)
```

The default `maxdsiz` on `11i v2` is 1 GB. This will make `procl` very slow in reaching its limits. You can change `maxdsiz` to a more reasonable number for this lab exercise by:

```
# ktune maxdsiz=0x10000000
```

WARNING: The automatic 'backup' configuration currently contains the configuration that was in use before the last reboot of this system.

==> Do you wish to update it to contain the current configuration before making the requested change? n

NOTE: The backup will not be updated.

* The requested changes have been applied to the currently running system.

Tunable		Value	Expression	Changes
maxdsiz	(before)	1073741824	Default	Immed
	(now)	0x10000000	0x10000000	

Also take some `vmstat` reading to satisfy yourself that the system is not under memory pressure. How much free memory do you have?

```
# vmstat 2 2
```

3. Use the **RUN** script to start the background processes:

```
# ./RUN
```

4. ● Open another window. Start **glance**. Sort the processes by CPU utilization (should be the default), and answer the following questions — fairly quickly, before the memory leaks get too large.
- What is the current amount of free memory? 21.16B
 - What is the size of the buffer cache? 1211B
 - Is there any paging to the swap space?
 - How much swap space is currently reserved?
 - Which process has the largest Resident Set Size (RSS)?
 - What is the data segment size of the process with the largest RSS?
5. After a several minutes, the **proc1** process should reach its maximum data size. If your **maxdsiz** is set to 1 GB, this could take a while. Please be patient. Observe the behavior of the system when this occurs.
- What happens when the process reaches its maximum data size?
 - Why does disk utilization become so high at this point?
6. As the other processes grow towards their maximum data segment size, continue to monitor the following:
- Free memory
 - Swap space reserved
 - The size of the processes' data segments
 - The RSS of the processes
 - The number of page-outs/page-ins to the swap space

Module 6

Memory Management

7. Run the two baseline programs, **short** and **diskread**.

```
# timex /home/h4262/baseline/short
# timex /home/h4262/baseline/diskread
```

How does the performance of these programs compare to their earlier runs?

8. When finished monitoring the behavior of processes with memory leaks, clean up the processes.

- Exit glance.
- Execute the **KILLIT** script:

```
# ./KILLIT
```

- If you changed **maxdsiz**, change it back:

```
# kctune maxdsiz=0x40000000
```

WARNING: The automatic 'backup' configuration currently contains the configuration that was in use before the last reboot of this system.

==> Do you wish to update it to contain the current configuration before making the requested change? n

NOTE: The backup will not be updated.

- * The requested changes have been applied to the currently running system.

Tunable		Value	Expression	Changes
maxdsiz	(before)	0x10000000	0x10000000	Immed
	(now)	0x40000000	0x40000000	

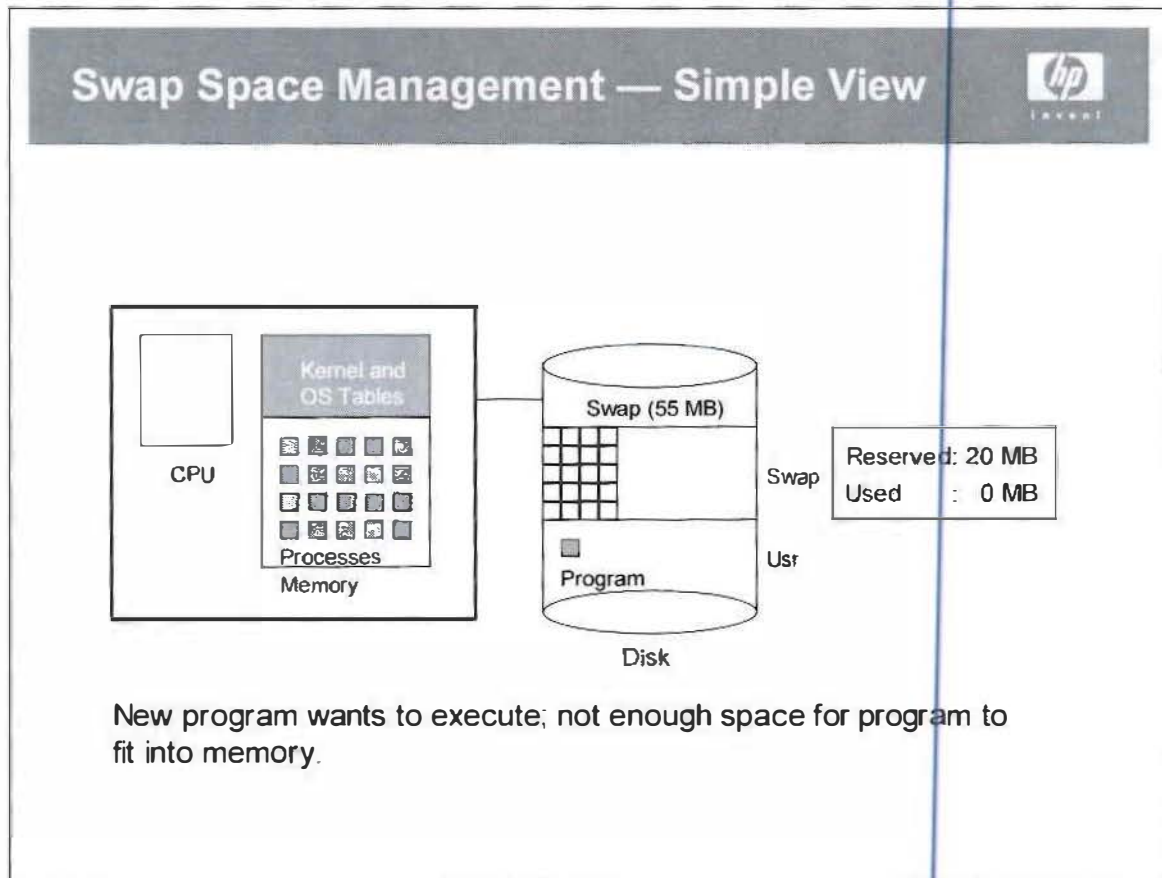
Module 7 — Swap Space Performance (proper & HP)

Objectives

Upon completion of this module, you will be able to do the following:

- Describe the difference between swap usage and swap reservation.
- Interpret the output of the `swapinfo` command.
- Define and configure pseudo swap.
- Define and configure swap space priorities.
- Define and configure `swchunk` and `maxswapchunks`.

7-1. SLIDE: Swap Space Management — Simple View



Student Notes

The purpose of “swap space” is to relieve the pressure on memory when memory becomes too full. When “free memory” falls below a certain threshold, processes (or parts of processes) will be written out to the swap partition on disk in order to free up space in memory for other processes.

For simplicity, the above slide assumes each process is 1 MB in size, and the amount of available memory for process execution is 20 MB. The slide also assumes (for simplicity) that each process “reserves” 1-MB on the swap partition each time it executes. Therefore, since 20 processes are currently present in memory (as shown on the slide), 20 MB of swap space has been reserved—1 MB for each process.

The IIP-UX operating system “reserves” swap space for each process that executes on the system. The reservation of swap space is done so that the operating system knows how much swap space “potentially” may be needed for all the processes currently running on the system. For example, if all the processes in memory were to be swapped out, the operating system would know it had enough swap space to perform that function.

Analogy

A good analogy for swap space reservation, is a hotel that takes room reservations. When a hotel takes a reservation, it subtracts one from the count of available rooms. If a hotel had 55 rooms, and it took 20 reservations, then it would only have 35 rooms still available, even though none of the 55 rooms were currently occupied.

The same holds true for swap space. In the above example, a total of 55 MB of swap space exists, 20 MB of the space is "reserved" by processes currently running in memory, even though none of the processes are currently using the swap space they have reserved.

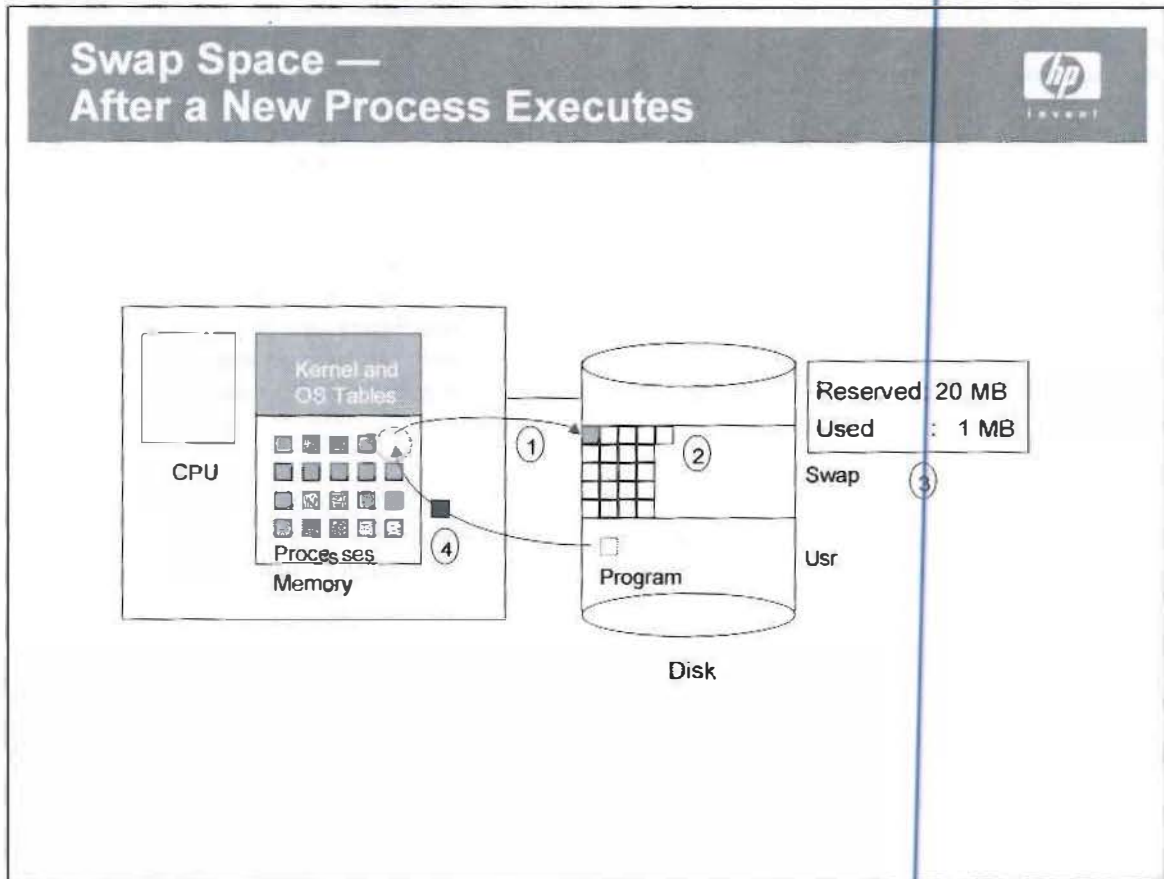
To take the analogy even further, the hotel does not earmark a particular room to satisfy a reservation. Room assignments are done when the occupant shows up at the front desk. Likewise, a swap reservation is not associated with a particular block out on the swap device. Only when the kernel actually wants to move a page in memory out to the swap device does it select a block. It knows it has the swap space available. It just doesn't know where it is until it needs to use it.

Current Situation

In the above slide, all the memory is in use by the 20 processes. Now assume a new program from disk wants to execute. What happens? How does it fit in memory if all the memory is in use?

la swap est une conteneur de la RAM.
Un processeur est allouer sur la swap avant de monter en
memore.

7-2. SLIDE: Swap Space — After a New Process Executes



Student Notes

Below is the basic sequence of steps that occurs when a new process wants to execute and there is not enough memory available:

1. The operating system selects a process (or portion of a process) to be written out to the swap partition on disk. The process selected is one that is not expected to execute in the near future.
2. Once the process is written to the swap partition, the amount of swap space used is incremented accordingly and the amount of swap space reserved is decremented by the same amount.
3. The new program which wants to execute reserves swap space for itself. The amount of swap space reserved is incremented accordingly.
4. The new program is copied into memory and the operating system initializes the process. The new process uses the physical memory that was just freed.

7-3. SLIDE: The swapinfo Command

The swapinfo Command									
									
# swapinfo -mt									
	Mb	Mb	Mb	PCT	START/	Mb			
TYPE	AVAIL	USED	FREE	USED	LIMIT	RESERVE	PRI	NAME	
dev	32	1	31	3%	0	-	1	/dev/vg00/lvol2	(
localfs	23	0	23	-	none	0	1	/home/paging	)
reserve	-	20	-20						
total	55	21	34	38%	-	0	-		

Student Notes

The **swapinfo** command displays important swap-related information, including how much swap space is *used*, and how much swap space is *reserved*. With today's systems, we recommend that you always use the **-m** option to display all spaces in MB rather than the default KB.

The **swapinfo -mt** command shows information related to *device* (raw) swap partitions and *file system* swap space and their totals, including:

Mb AVAIL	The total amount of swap space available. For file system swap, this value may vary, as more swap space is needed.
Mb USED	The current amount of swap space being used.
Mb FREE	The current amount of swap space free. The Mb FREE plus Mb USED is equal to Mb AVAIL.
PCT USED	The percentage of swap space in use on that device.

Module 7

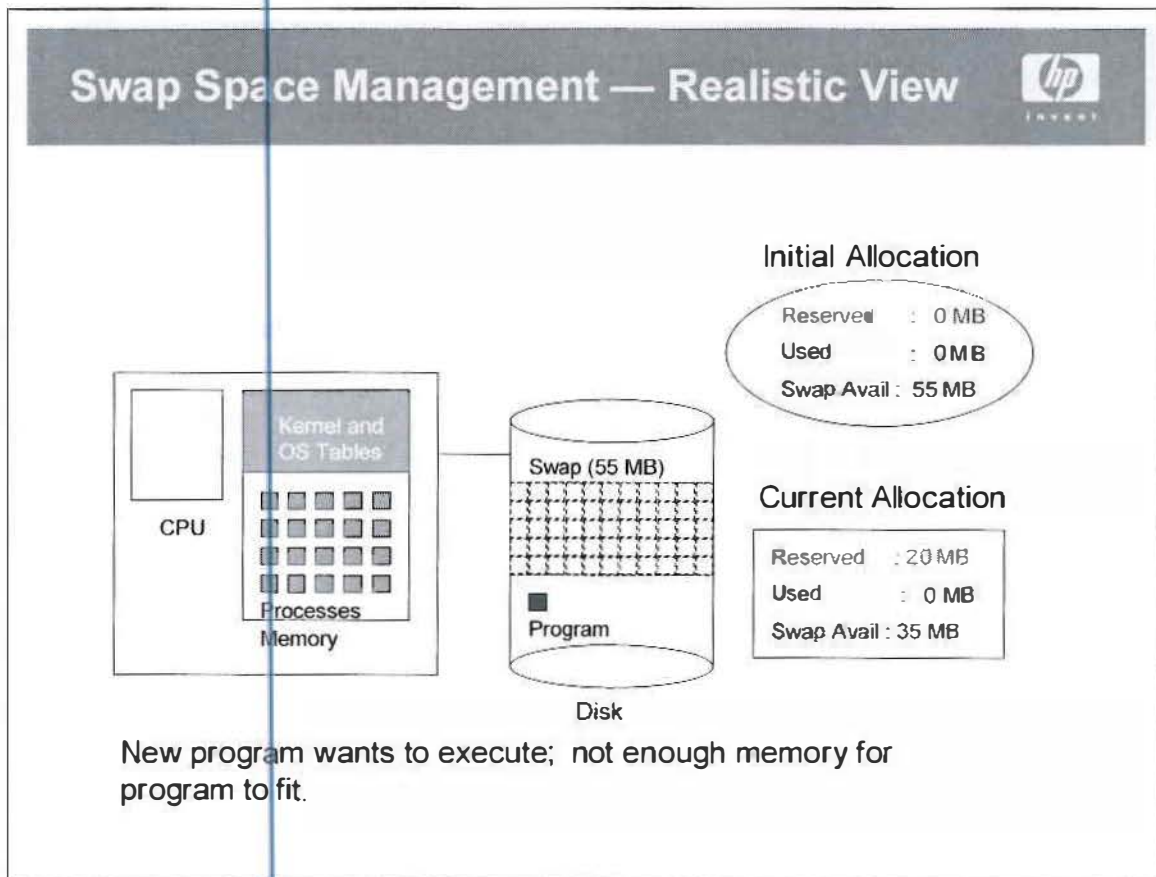
Swap Space Performance

START/LIMIT	Applies only to file system swap. Specifies the starting block within the file system of the paging file. The LIMIT specifies the maximum size to which the paging file can grow.
Mb RESERVED	Applies only to file system swap, and is only applicable when no limit is given to the maximum size of the paging file. In these situations, this value specifies how much file system space to reserve for user files on the file system.
PRI	The priority of the swap area. The highest priority swap areas are used first. The swap priorities range from 0-10. (Note: stronger priority swap areas have smaller priority numbers.)

The **swapinfo** command also shows how much swap space all the processes on the system are reserving currently. This is indicated by the **reserve** entry. The columns described above for device and file system swap do not apply to the **reserve** entry in the output of the **swapinfo** command.

In the example, there are 32 MB of device swap on a raw disk, and 23 MB of swap in the **/home** file system, making a total of 55 MB. 1 MB is in use on the device swap and 20 MB are reserved, leaving 34 MB available.

7-4. SLIDE: Swap Space Management — Realistic View



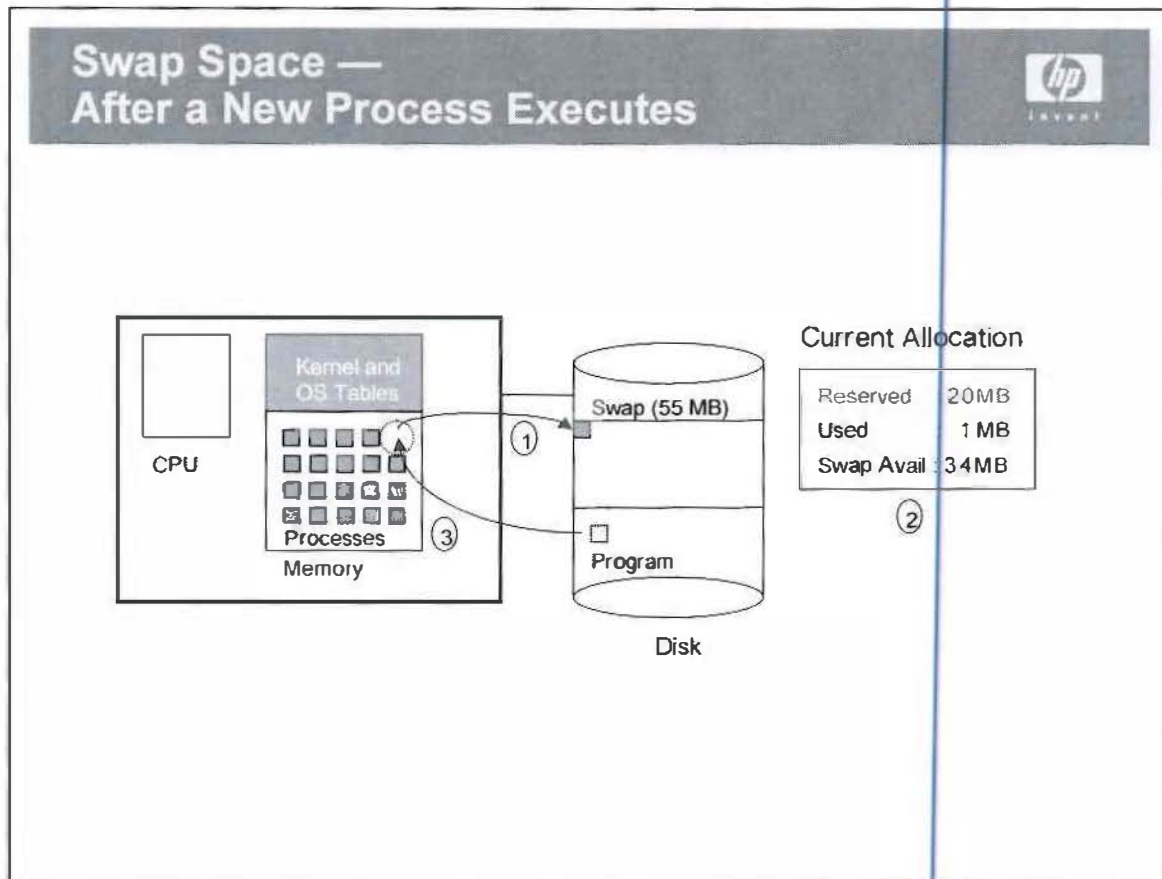
Student Notes

An earlier slide implied that specific space was allocated on a swap device for each process running in memory. The analogy was of a hotel subtracting one from the count of available rooms when a customer phoned in for a reservation.

As mentioned earlier, specific space is *not* allocated on a swap device for a reservation. Instead, a variable is maintained called **SWAP_AVAIL**. The **SWAP_AVAIL** variable is initialized when the system boots to equal the total amount of swap space available. As each new process begins executing, this variable is decremented according to the amount of swap space the process would need if its entire contents were to be swapped out. When a process terminates, it returns the amount of swap space it reserved back to the **SWAP_AVAIL** variable.

The slide above shows what the **SWAP_AVAIL** variable would contain when 20 MB worth of processes is executing on the system. Each process has caused the **SWAP_AVAIL** variable to be decremented, but no specific space has been allocated on the swap partition. No specific swap space is allocated until processes need to be paged out, as shown on the next slide.

7-5. SLIDE: Swap Space — After a New Process Executes



Student Notes

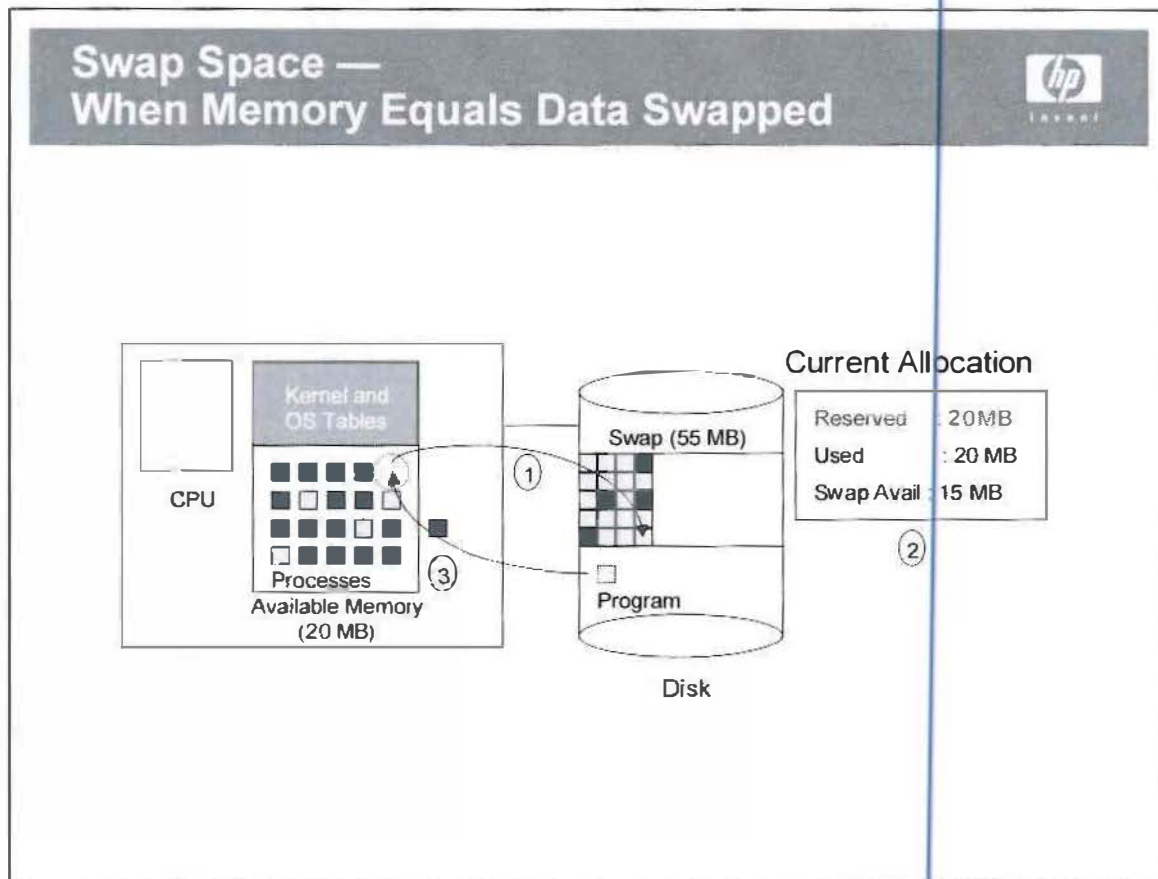
This is an updated description of the sequence of events that occurs when a program is being executed and not enough memory is available:

- The operating system selects a process (or portion of a process) to be written out to the swap partition on disk. Since no specific swap space has been reserved, swap space is allocated from the strongest priority swap device, first available block.
- Once the process is written to the swap partition, the amount of swap space *used* is incremented accordingly, and the old program “unreserves” its swap space by incrementing the `SWAP_AVAIL` variable.
- Then the new program decrements `SWAP_AVAIL` to reserve its swap space. In effect, the amount of swap space *reserved* is decremented by the amount of space being moved out to swap space and then incremented by the new reservation amount.

In the slide, the process being swapped out causes the `USED` swap to become 1 MB, causing the `SWAP_AVAIL` to become 34 MB. Then the old process releases its 1 MB reservation, causing the `SWAP_AVAIL` to increase back to 35 MB. Finally, the new process starts up and causes the `SWAP_AVAIL` to decrease from 35 to 34 MB.

- The new program is copied into memory, and the operating system initializes the process after it has confirmed that it can successfully reserve the needed swap for the new process (SWAP_AVAIL does not go negative when the swap reservation is made).

7-6. SLIDE: Swap Space — When Memory Equals Data Swapped



Student Notes

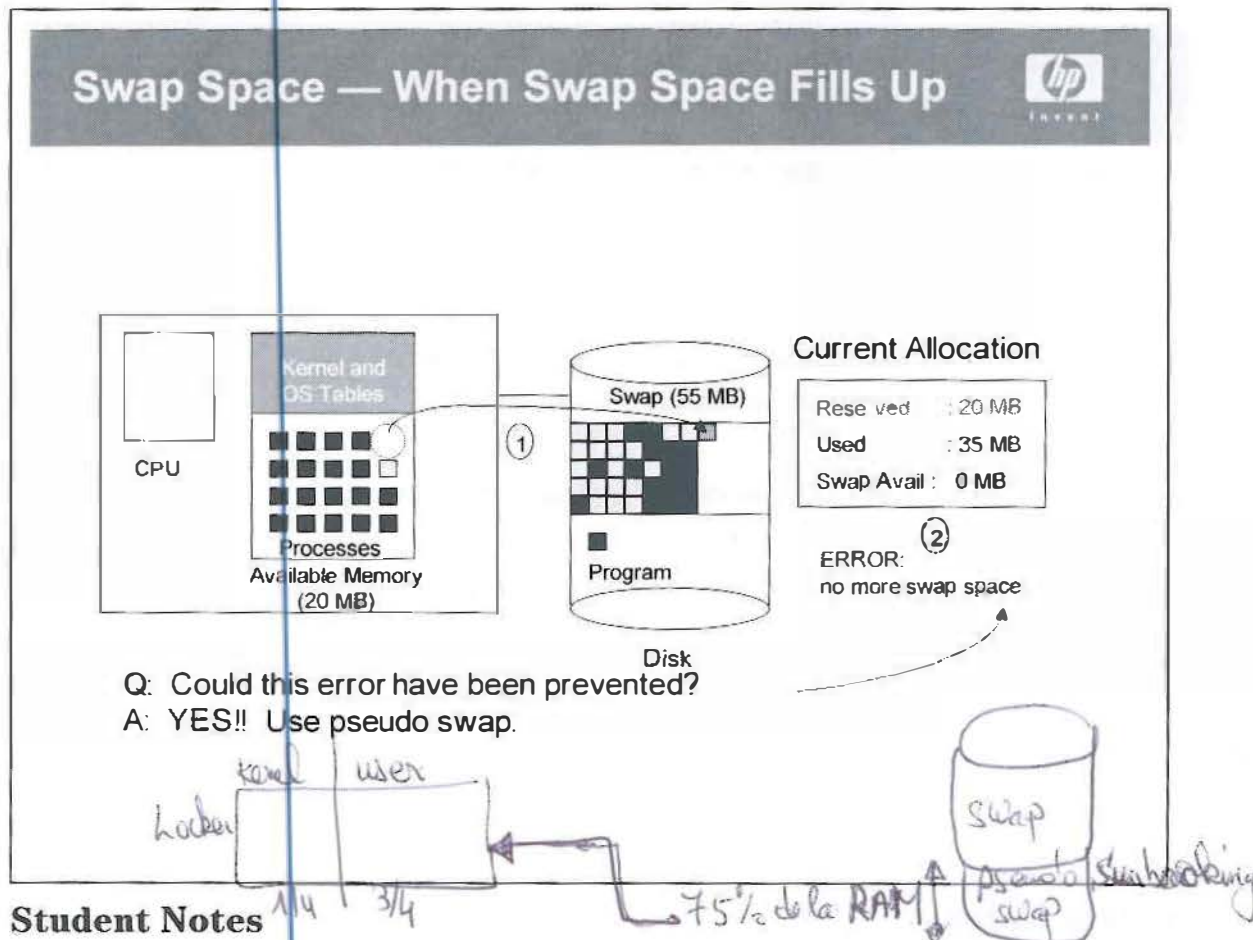
The above slide shows the state of the system and the current swap space allocations when 20 MB (or all of available memory) has been paged out to the swap partition.

The swap partition contains 20 MB worth of processes, which is the size of available memory. The initial 20 MB of processes is shaded in gray, to distinguish them from the second 20 MB of processes, which are filled with black. With this color code, we can see only 4 MB of the original processes are still loaded in memory, everything else (including 4 MB of the 21st to 40th processes) has been paged to the swap partition.

The swap space allocation reflects 20 MB worth of processes that have *reserved* swap space, and 20 MB that is currently in use. This would be analogous to stating that a hotel received 40 room reservations, and 20 of those reservations are currently being used.

The `SWAP_AVAIL` variable is down to 15 MB, because the total amount of swap space is 55 MB and 40 MB of that space is reserved or in use.

7-7. SLIDE: Swap Space — When Swap Space Fills Up



Student Notes

The above slide shows the situation when `SWAP_AVAIL` equals 0 MB. In this situation, the error message, **ERROR: no swap space available** is displayed, even though there is swap space to page an existing process to the swap partition and thus free up memory for a new program to load.

The reason the system reports no swap space is available is because 35 MB of memory have been paged out, and the remaining 20 MB of swap space are reserved by the existing processes currently executing in memory.

Could this error have been prevented?

From a resource perspective, the new program should be able to execute, because memory is available for the new process.

A tunable OS parameter, referred to as **pseudo swap**, would have allowed the program to execute under these conditions.

7-8. SLIDE: Pseudo Swap

Pseudo Swap		
Definition:	Pseudo swap is fictitious, make-believe, swap space. It does NOT exist physically, but logically the operating system recognizes it.	
Purpose:	Pseudo swap allows more swap space to be made available than physically exists.	
Benefit:	Pseudo swap adds "75% of physical memory" to the amount of swap space that the operating system thinks is available. This lessens swap space requirements (especially helpful on large memory systems.)	
**NOTE:	Pseudo swap is NOT allocated in memory!	

Student Notes

Pseudo swap is HP's solution for large memory customers who do not wish to purchase a large amount of disks to use for swap space. The justification for purchasing large memory systems is to prevent paging and swapping, therefore, the argument becomes, "Why purchase a lot of device swap space if the system is not expected to page or swap?"

Pseudo swap is swap space that the operating system recognizes, but in reality it does not exist. Pseudo swap is make-believe swap space. It does *not* exist in memory; it does *not* exist on disk; it does *not* exist anywhere. However, the operating system does recognize it, which means more swap space can be reserved than physically exists.

The purpose of pseudo swap is to allow more processes to run in memory than could be supported by the swap device(s). It allows the operating system (specifically the **SWAP_AVAIL** variable) to recognize more swap space, thereby allowing additional processes to start when all physical swap has been reserved. By having the operating system recognize more swap space than physically exists, large memory customers can now operate without having to purchase large amounts of swap space, which they will most likely never use.

The size of pseudo swap is dependent on the amount of memory in the system. Specifically, the size is (approximately) 75% of physical memory. This means the **SWAP_AVAIL** variable

will have an additional amount (75% of physical memory) added to its content. This additional amount allows more processes to start when the physical swap has been completely reserved.

NOTE: Pseudo swap is enabled through a tunable OS parameter call `swapmem_on`.

If the value for `swapmem_on` is 1, then pseudo swap will be enabled (turned on).

If the value for `swapmem_on` is 0, then pseudo swap will be disabled (turned off).

Analogy

A good analogy for pseudo swap is an airline overbooking a flight. Airlines know that customers sometimes don't show up for their flight. If they reserved only enough seats for the plane, they would likely depart with a plane that wasn't full – lost revenue. So they reserve more seats than actually exist on the plane, betting that a certain percentage of customers won't show. That way they can fly a plane that is much closer to full and get more revenue. Of course, they are occasionally wrong.

7-9. SLIDE: Total Swap Space Calculation — with Pseudo Swap

Total Swap Space Calculation —
with Pseudo Swap



Memory Size	=	32 MB
x		0.75
Pseudo Swap	=	24 MB
+ Physical Swap	=	55 MB
Total Swap	=	79 MB

Student Notes

The above slide shows how “Total Available Swap Space” (also known as `SWAP_AVAIL`) is calculated with pseudo swap turned on.

The `SWAP_AVAIL` variable is calculated as all of the configured physical swap space (device and file system swap) PLUS 75% of physical memory (pseudo swap).

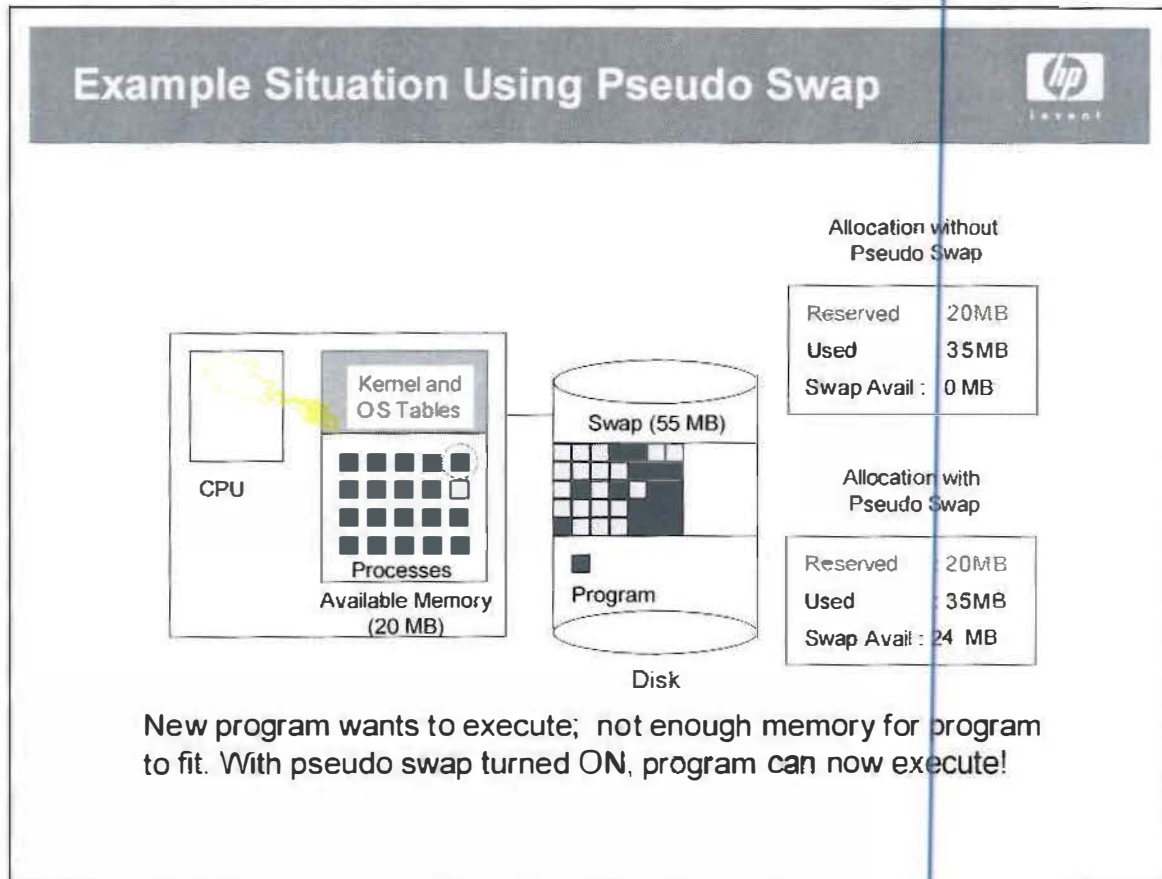
(The calculation of the size of pseudo swap is actually more complex than given here. The resultant value of pseudo swap can vary anywhere from 67% to 88% of physical memory. But we'll use 75% as a pretty typical figure.)

In our example, the total amount of physical swap was 55 MB, and the amount of physical memory was 32 MB. Since the size of pseudo swap is estimated at 75% of physical memory, the pseudo swap size in our example is 24 MB.

This means the Total Available Swap Space (SWAP_AVAIL) is:

```
55 MB (Physical Swap)
+ 24 MB (Pseudo Swap)
-----
79 MB (Total Avail Swap)
```

7-10. SLIDE: Example Situation Using Pseudo Swap



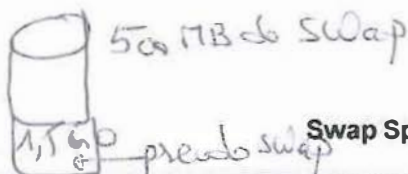
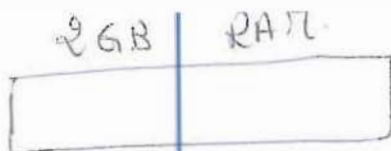
Student Notes

The above slide revisits our previous situation with pseudo swap turned ON. In our previous situation, we had swap space of 55 MB, of which 35 MB was in use and the remaining 20-MB was reserved. With pseudo swap turned OFF, we saw that no new processes could start because no physical swap space was available for reservation purposes.

With pseudo swap turned ON, the total available swap space is 79 MB (not 55 MB). Therefore, when the system runs out of physical swap, it still has 24 MB (due to pseudo swap), which it thinks it can allocate and therefore can reserve.

Consequently, the operating system is able to support more processes without having to allocate more physical swap space. This is important for large memory customers who do not want to purchase a lot of swap space on disk in order to support the large memory.

3)



Module 7

Swap Space Performance

7-11. SLIDE: Swap Priorities

si j'ai pas de swap dump et us d'a fait
sur la swap de boot

Swap Priorities

Equal Priorities

1st chunk of swap - disk 1, chunk 1
 2nd " " disk 2, chunk 1
 3rd " " disk 1, chunk 2
 4th " " disk 2, chunk 2
 5th chunk will be allocated here

1
3

2
4

Swap - Priority 1 Swap - Priority 1

Unequal Priorities

1st chunk of swap - disk 1, chunk 1
 2nd " " disk 1, chunk 2
 3rd " " disk 1, chunk 3
 4th " " disk 1, chunk 4
 5th chunk will be allocated here

1 4
2
3

1
2
3

Swap - Priority 1 Swap - Priority 2

1) de preference il faut avoir la meme taille des disk swap et avec une
priorite pour eviter le calcul sur CPU ou bien si vous avez
deux disk

Student Notes

When the HP-UX operating system needs to page something from memory to a swap device, it selects the smallest-numbered, strongest-priority swap device.

A system administrator can define a priority number for each swap device on the system. The priority numbers range from 0 to 10, with 0 being the strongest priority, and 10 being the weakest priority. If multiple swap devices are available when the system needs to page out to swap, the strongest priority swap device is used.

The slide shows two examples. The first example illustrates how the system behaves when two *equal* priority swap devices are available. In this situation, the system alternates between the two swap devices, with the first chunk of swap being allocated on swap device #1, and the second chunk of swap being allocated on swap device #2.

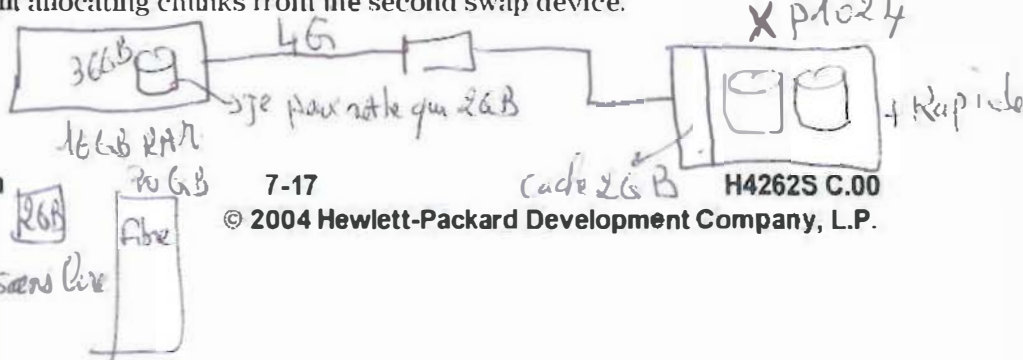
The second example illustrates how the system behaves when two *unequal* priority swap devices are available. In this situation, the system will continue to allocate chunks of swap from the lowest-numbered (strongest priority) swap device. Only when that device is 100% full will the system begin allocating chunks from the second swap device.

<http://education.hp.com>

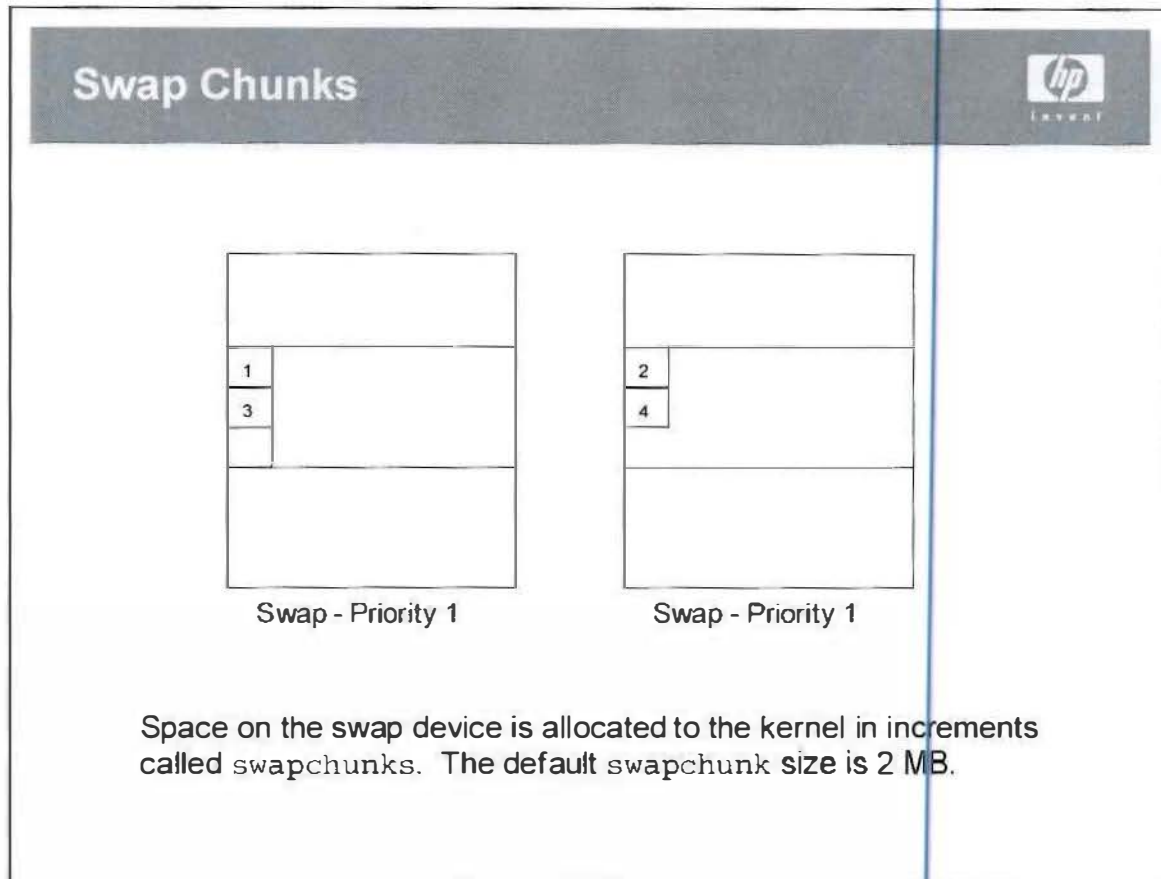
7-17

© 2004 Hewlett-Packard Development Company, L.P.

donc prendre 2GB avec
LVM boot qui va demarrer sans lire
le fstab



7-12. SLIDE: Swap Chunks



Student Notes

A swap chunk is the amount of space that the operating system allocates swap devices. The default swap chunk size is 2 MB.

In the above example, two equal priority swap devices are available to the system. The system will allocate the first swap chunk to be on swap device #1, and this size will be 2 MB by default. Once this swap chunk has been filled by 512 pages (page size = 4 KB), then the system will allocate a second swap chunk to be on swap device #2. The system continues alternating swap space between the two systems in swap chunk increments.

Swap chunks are also the unit in which swap space is allocated on file system swap devices. With file system swap devices, the operating system will only allocate swap space on the file system if the space is needed; if it does not need the swap space, then it does not allocate space. When it does need swap space, it allocates the file system swap space in *swap chunk* sizes. Files are created – each of a size equal to a swap chunk – and named *h_{ostname}mcN*, where N is a number from 0 on up.

7-13. SLIDE: Swap Space Parameters

Swap Space Parameters


DEV_BSIZE	Device block size. This is the size (in bytes) of a block on the disk. The default size is 1024 bytes.
swchunk	This is the number of blocks to allocate to the kernel when it need swap space. The default is to allocate swap space to the kernel in 2-MB increments. The default value is 2048. The maximum value is 65,536.
maxswapchunks	This is the maximum number of swchunks which can be allocated to the kernel. The default value is 256. The maximum value is 16,384.

Total swap space recognized by the kernel =

$$\text{maxswapchunks} \times \text{swchunk} \times \text{DEV_BSIZE}$$

Defaults: 256 x 2048 x 1024 = 512 MB

*à définir avec
grand valeurs
au début*

Student Notes

There are two configurable parameters and one fixed, non-configurable parameter that affect swap space configurations and allocations.

DEV_BSIZE	The size in bytes of a block of disk space. The default size is 1 KB. It is not configurable.
swchunk	The number of blocks (of size DEV_BSIZE) to associate with a "chunk" of swap space, referred to as a swap chunk . The default value is 2048 blocks or 2 MB. The maximum value is 65,536 or 64 MB.
maxswapchunks	This is the maximum number of swap chunks that will be recognized systemwide. The default value is 256. The maximum value is 16,384.

Using these defaults, the maximum amount of swap space that the operating system recognizes is 512 MB. This means if a system is configured physically for 1 GB of swap space, only 512 MB of the 1 GB will be used by the system. In order for the system to use the other 512 MB, the tunable OS parameter **maxswapchunks** needs to be increased to 512.

Module 7

Swap Space Performance

If you were to install HP-UX on a system that had 2 GB of physical memory, the installation process would automatically increase `maxswapchunks` to accommodate the larger memory. In this example, it would set `maxswapchunks` to 1024. However, if you were to add more memory at a later date (without reinstalling the kernel), you would have to manually tune `maxswapchunks` to be able to allocate enough swap space and use all of your available memory. Or, use pseudo swap.

In 11.23 (11i v2), `maxswapchunks` has been eliminated and no longer becomes an issue.

7-14. SLIDE: Summary

Summary



- Swap space reservation : un processeur réserve de la swap avant de monter en zéro
- Pseudo swap : 75% de la RAM
- Swap priorities :
- Swap chunks :
- Swap space parameters :

Swapinfo - atm

dev = (swap area) / dev / g / o / l / v / o / l / e

reserve = l'espace de la swap

memory = l'espace de la pseudo swap

Student Notes

To summarize this module, all processes must reserve swap space by decrementing a variable called `SWAP_AVAIL` when they initialize. If this variable cannot be decremented, the process will not be able to start.

To allow this variable to recognize more swap space than physically exists, setting a tunable parameter, `swapon_0n`, to 1 will turn on pseudo swap. This allows more processes to execute than the amount of swap space can support. This is not considered a problem on large memory systems, because these machines are not expected to swap.

If a system does need to swap, it will swap to the lowest-numbered (strongest) priority swap device first. The priority of a swap device is specified when the device is activated. If two swap devices have the same priority, the system will alternate between the two devices.

Swap chunks are the unit of disk space by which swap space is allocated. By default, the size of a swap chunk is 2 MB. By default, the system recognizes a maximum of 512 MB of swap space. If more swap space exists, the tunable parameter, `maxswapchunks`, must be increased, in order for the additional swap space to be recognized. If `maxswapchunks` is already set to the maximum value, then increase the value of `swchunk`.

7-15. LAB: Monitoring Swap Space

Preliminary Steps

A portion of this lab requires you to interact with the ISL and boot menus, which can only be accomplished via a console login. If you are using remote lab equipment, access your system's console interface via the GSP/MP.

You may get some "file system full" messages while you are shutting down the system. You can ignore these messages.

Directions

The following lab illustrates swap reservation, configures and de-configures pseudo swap, and adds additional swap partitions with different swap priorities.

1. Use the `swapinfo` command to display the current swap space statistics on the system. List the MB Avail and MB Used for the following three items:

	MB Available	MB Used
dev	4096	941
reserve	-	227
memory	6132	1027

2. To see total swap space available and total swap space reserved, enter:

```
# swapinfo -mt
```

What is the total swap space available (including pseudo swap)? 6132

What is the total space reserved? 1027

3. Start a new shell process by typing `sh`. Re-execute the `swapinfo` command and verify whether any additional swap space was reserved when the new shell process started. In this case, the difference is going to be pretty small, so let's not use the `-m` option.

Upon verification, exit the shell.

Is the swap space returned upon exiting the shell process?

4. Start `glance` and observe the Global bars at the top of the display for the duration of this step. Start a large, memory process and note how much the Current Swap Utilization percentage increases in `glance`. Type:

```
# /home/h4262/memory/paging/mem256 &
```

Use the process that most closely matches your physical memory size. This should reserve a large amount of swap space. Start as many `mem256` processes as possible. For best results, wait until each swap reservation is complete, by observing the incremental increases in Current Swap Utilization in `glance`. The system will get slower and slower as you start more `mem256` processes.

What was the maximum number of `mem256` processes that can be started?

What prevented an additional `mem256` process from being started?

Kill all `mem256` processes to restore performance.

5. Recompile the kernel, disabling pseudo swap. Use the following procedure:

11i v1 or earlier

```
# cd /stand/build
# /usr/sbin/sysadm/system_prep -s system
# echo "swapmem_on 0" >> system
# mk_kernel -s ./system
# cd /
# shutdown -ry 0
```

Module 7

Swap Space Performance

11i v2 and later:

```
# cd /
# kctune swapmem_on=0
NOTE: The configuration being loaded contains the following
change(s)
    that cannot be applied immediately and which will be held for
    the next boot:
    -- The tunable swapmem_on cannot be changed in a dynamic fashion.
WARNING: The automatic 'backup' configuration currently contains the
configuration that was in use before the last reboot of this
system.
==> Do you wish to update it to contain the current configuration
before making the requested change? no
NOTE: The backup will not be updated.
    * The requested changes have been saved, and will take effect at
    next boot.

Tunable          Value Expression
swapmem_on (now)      1 Default
              (next boot) 0 0
# shutdown -ry 0
```

6. Reboot from the new kernel.

```
Press any key to interrupt the boot process
Main menu> boot pri isl
Interact with IPL> y
ISL> hpx (;0)/stand/build/vmunix_test
```

7. Once the system reboots, login and execute `swapinfo`.

Is there a memory entry? Why or why not?

Will the same number of `mem256` processes be able to execute as earlier?

How many `mem256` processes can be started now?

Kill all `mem256` processes to restore performance.

8. If you have a two disk system.

If you have a two disk system, add the second disk to `vg00` (if this was not already done in a previous exercise) and build a second swap logical volume on it. This `lv01` should be the same size as the primary swap volume. **If you do not have a second disk continue this lab at question 13.**

If you did not add the second disk earlier:

```
# vdisplay -v | grep Name (Note the physical disks used by vg00)
# iocan -fnc disk (Note which disks are unused)
# pvcreate -f <raw_dev_file_of_unused_disk>
# vextend /dev/vg00 <block_dev_file_of_second_disk>
```

To create the new swap device on the second disk:

```
# lvcreate -n swap1 /dev/vg00
# lvextend -L 512 /dev/vg00/swap1 <dev_file_of_second_disk>
```

Note in our case the primary swap is 512MB. See **swapinfo** on your system and match the size of the new swap device to the primary swap.

9. Now add the new logical volume to swap space. Ensure that the priority is the same as the primary swap. Check your work.

```
# swapon -p 1 /dev/vg00/swap1
swapon: Device /dev/vg00/swap1 contains a file system.
Use -e to page after the end of the file system, or -f to
overwrite the file system with paging.
```

Oops! Problem 1, swapon is being overly cautious. If you get this message, the memory manager has detected what appears to be a file system already on the device. (Probably, left over from some previous use.) You need to override.

```
# swapon -p 1 -f /dev/vg00/swap1
swapon: The kernel tunable parameter "maxswapchunks" needs to
be increased to add paging on device /dev/vg00/swap1.
```

Oops! Problem 2, the kernel cannot deal with this amount of swap. If you get this message, the tunable parameter, **maxswapchunks**, is set too small to accommodate all of the new swap space. We need to modify "maxswapchunks" and reboot. If you have this problem, use **sam** to double **maxswapchunks**. In **11i v2**, **maxswapchunks** has been obsoleted and will not have to be modified.

Recompile the kernel, increasing **maxswapchunks**. Use the following procedure:

```
# cd /stand/build
# echo "maxswapchunks 512" >> system
# mk_kernel -s ./system
# cd /
# shutdown -ry 0
```

10. If you had to rebuild the kernel to increase **maxswapchunks**, reboot the system. Otherwise, skip to step 11.

```
Press any key to interrupt the boot process
Main menu> boot pri isl
Interact with IPL> y
ISL> hpux (;0)/stand/build/vmunix_test
```


Module 7
Swap Space Performance

And now add the new swap device:

```
# swapon -p 1 -f /dev/vg00/swap1
```

Verify that the new swap space has been recognized by the kernel:

```
# swapinfo -mt
```

Done!

11. Start enough memory processes to make the system start paging.
12. Measure the disk I/O to see what is happening with swap space. **Go to question 15 when you have finished.**
13. **If you have a single disk system.**

Create three additional swap devices with sizes of 20 MB.

```
# lvcreate -L 20 -n swap1 vg00  
# lvcreate -L 20 -n swap2 vg00  
# lvcreate -L 20 -n swap3 vg00
```

List the current amount of swap space in use.

If 10 MB is currently in use on a single swap device, and we activate an equal priority swap device, what is the distribution if an additional 10 MB is paged out?

- A) The distribution would be 10 MB and 10 MB.
- B) The distribution would be 15 MB and 5 MB.

Prior to activating these swap devices, make note of the amount of swap space currently in use. When the new swap devices are activated with equal priority, all new paging activity will be spread evenly over these swap devices.

14. Activate the newly created swap devices. Activate two with a priority of 1, and the third with a priority of 2.

```
# swapon -p 1 /dev/vg00/swap1
# swapon -p 2 /dev/vg00/swap2
# swapon -p 1 /dev/vg00/swap3
```

Start enough ~~mem~~256 processes to make the system start paging.

Is the new paging activity being distributed evenly across the paging devices?

15. When finished with the lab, **reboot** the system as normal (do not boot `vmunix_test`) to re-enable pseudo swap and remove the additional swap devices.

For 11i v1 and earlier, follow this procedure:

```
# cd /
# shutdown -ry 0
```

For 11i v2 and later, follow this procedure:

```
# cd /
# kctune swapmem_on=1
# shutdown -ry 0
```

Module 7
Swap Space Performance

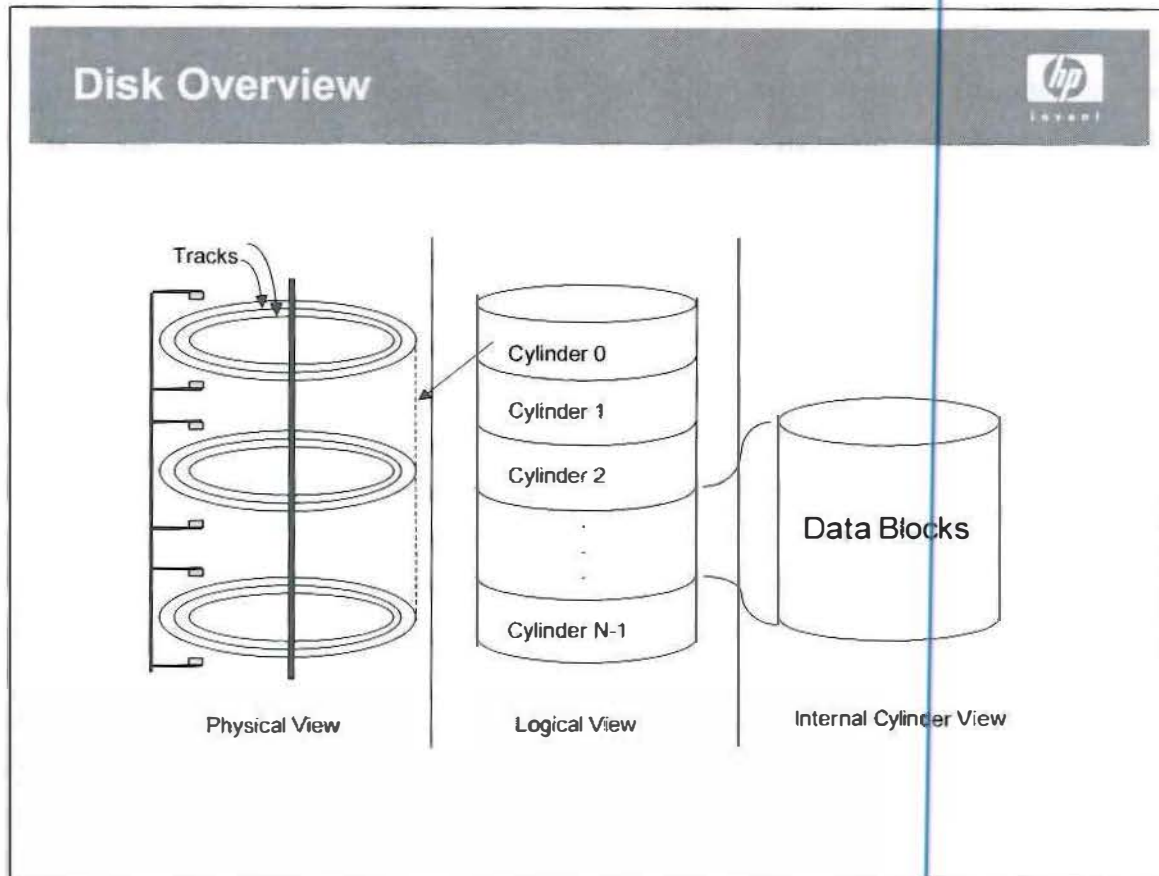
Module 8 — Disk Performance Issues

Objectives

Upon completion of this module, you will be able to do the following:

- List three ways disk space can be used.
- List disk device files.
- Identify disk bottlenecks.
- Identify kernel system parameters.

8-1. SLIDE: Disk Overview



Student Notes

Disks are used to store data for the operating system and the applications. A disk can be used several different ways, but they boil down to just two – file system and raw. If a disk holds a file system, there are several structures which are built on the disk (using the data blocks of the disk) to help support the software in the kernel, which needs to access and manage the file system files and their contents.

If the disks are to be used raw (such as a device swap space or an application database), no kernel structures are built out on the disk. The related code simply reads, manages and organizes the data blocks as it sees fit.

There are several types of file systems available with the HP-UX 10.x and 11.x releases. The two primary types of local file systems are HFS (High performance File System), which was the original file system for HP-UX and has continually been enhanced since, and JFS (Journaled File System), which was introduced with the HP-UX 10.01 release and continues to grow in popularity and functionality.

In the near future, you should see another type of file system become available for HP-UX – the Advanced File System (AdvFS) ported over from Tru64 UNIX. In later modules, we will

discuss the performance issues that pertain to each of the available file systems. In this module, we'll address the issues pertaining to all disks.

Physical View

From a physical disk perspective, the disk drives upon which a file system is placed contains sectors, tracks, platters, and read/write heads.

A key behavior of most all disk drives is that the read/write heads move in parallel across the platters in such a way that each read/write head is over the same track within each platter at the same time. To maximize the I/O throughput of the disk, it is desirable to minimize the amount of head movement. To help achieve this goal, all the sectors in a cylinder are addressed in sequential order.

Cylinder Analogy

Consider a health spa or gym with three floors. Each floor contains a jogging track, and the three jogging tracks are located directly above or beneath one another from floor to floor. From this point of view, a cylinder would be all the same lanes from each floor's jogging track. In other words, all lane 1 tracks would make up cylinder 1; all lane 2 tracks would make up cylinder 2, etc.

By organizing space on disks in cylinders, the software can logically distribute its sectors across all platters of the disk evenly and uniformly. For example, in the slide above, the first 6 sectors would be allocated as follows:

block #1:	Platter #1, Track #1, Sector #1
block #2:	Platter #1, Track #1, Sector #2
block #3:	Platter #1, Track #1, Sector #3
block #4:	Platter #1, Track #1, Sector #4
block #5:	Platter #1, Track #1, Sector #5
block #6:	Platter #1, Track #1, Sector #6

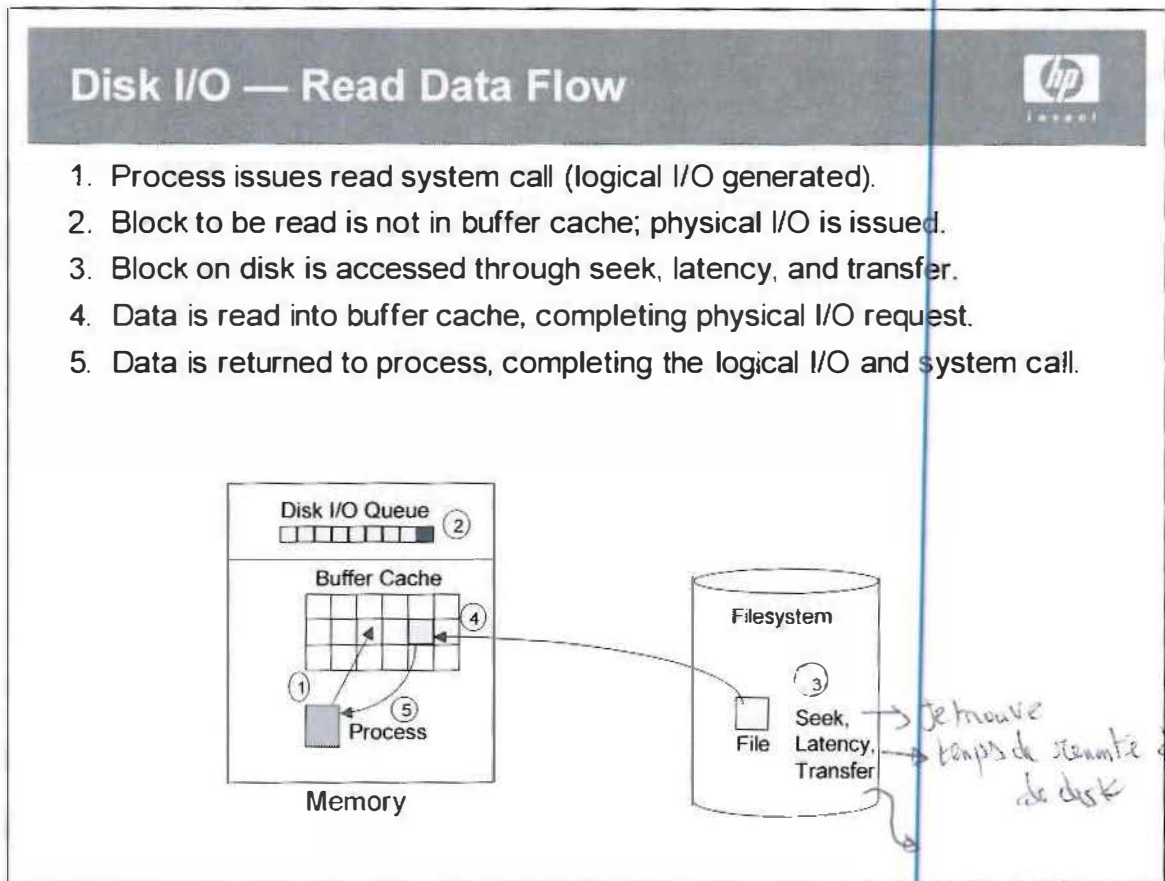
By allocating disk space in this manner, a multiple block read (say 6 blocks) could be read in one operation.

Logical View

From a logical view, each cylinder is simply a repository for a certain amount of data, which can be read or written without having to move the heads. This data area is further broken down into blocks. The block is the most fundamental unit of data that can be read from or written to the disk. We mentioned in an earlier chapter a value in the kernel, called `DEV_BSIZE`. It is equal to 1024 bytes. This is the block size from the kernel's perspective.

The disk can be viewed as simply a series of blocks running from block 0 to block N-1, where N is the total number of blocks on the disk. The closer two blocks are to each other, the more likely they will be in the same cylinder. If they are in the same cylinder, a minimum amount of time is needed to read or write both blocks.

8-2. SLIDE: Disk I/O — Read Data Flow



Student Notes

Up to this point, we have looked at I/O from the standpoint of the disk. The following slide illustrates disk I/O activities from the standpoint of memory and the process initiating the I/O. The assumption here is that we are dealing with a disk that has a file system on it, so the buffer cache becomes a factor in the operation. If this were a raw disk, the buffer cache would be bypassed by all I/O operations.

Asynchronous vs. Synchronous Reads

There are two possible approaches to doing reads – synchronous and asynchronous. By default any read will be synchronous, i.e., the process will wait (and sleep, if necessary) until the data can be transferred to the data area of the process.

If the read is asynchronous, the process informs another driver (an asynchronous I/O driver) that it will need certain data in the future. The driver fetches the data from the disk and places it in the buffer cache, while the process continues with other operations. When the data is in the cache, the driver signals the process and the read is now executed.

The data is guaranteed to be in the buffer and the process never has to sleep. Asynchronous reads are significantly more difficult to program, so they are used only in the more sophisticated applications.

Buffered Read Data Flow

The flow diagram on the slide highlights the main actions from the time a process issues a `read()` system call, to when the data is returned to the process.

1. A process issues the `read()` system call. This is viewed by the kernel as a **logical I/O**, meaning the kernel will satisfy the request any way it can, either through the buffer cache or by performing a **physical I/O**.
2. The buffer cache is searched, looking for the data blocks being requested. If the data block is found in the buffer cache, the `read()` system call is returned with the corresponding data. If the data block is not found, The requesting process goes to sleep and a physical I/O request is generated to read the data block into the buffer cache. We will assume the data block was not found.

NOTE: Logical I/Os may or may not generate corresponding physical I/Os. The goal of the buffer cache is to handle as many logical I/Os with as few physical I/Os as possible.

3. The physical read is performed because the data was not in the buffer cache. Because physical I/O involves movement of the disk head (**seek time**), waiting for the data on the platter to rotate under the disk head (**latency time**), and moving the data from the platter into memory (**transfer time**), the cost of a physical I/O is high from a performance standpoint. *Physical I/Os are the most time-consuming operations that the kernel performs.*

If the disk I/O queue is long (3 or more requests), the time spent *waiting* to be serviced can be longer than the time to actually service the I/O request.

4. Once the physical I/O request returns, the data is stored in the buffer cache so that future I/O requests for the same file system block can be satisfied without having to perform another physical I/O. This step completes the physical I/O initiated by the kernel.
5. The final step is to return the data to the original calling process that issued the `read()`. The sleeping process is awakened and transfers the desired data from the buffer (in buffer cache) to the data area of the process. Then the process returns from the `read()` system call. This step completes the logical I/O initiated by the process.

Raw Read Data Flow

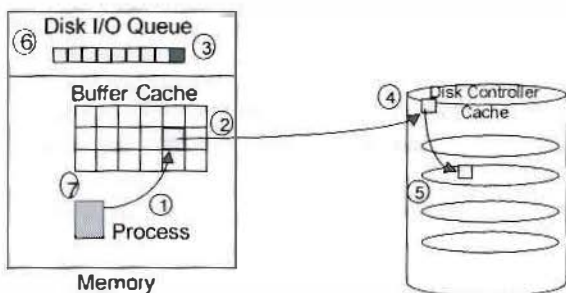
If the read operation is raw, the buffer cache is bypassed. Data is transferred directly from the disk to the data area of the calling process. All raw reads are synchronous and therefore result in the process sleeping until the data has been read in.

8-3. SLIDE: Disk I/O — Write Data Flow (Synchronous)

Disk I/O — Write Data Flow (Synchronous)



1. Process issues write system call.
2. Block is assigned on disk, and image for block is allocated in buffer cache.
3. Once data is written to buffer cache, a physical I/O to disk is generated.
4. Data is written to disk controller cache.
5. Data is then transferred from the disk controller to the corresponding platter.
6. Upon completion of I/O, the disk controller sends an acknowledgment to the kernel.
7. Write system call returns to process.



Student Notes

As with reads, there are two methods for performing `write()` system calls: asynchronous and synchronous. Although the default write operation is asynchronous (the writing process does not sleep – waiting for the write to complete), it is quite simple for a program to choose synchronous writes. It can be done by simply setting a flag on the open file before issuing the write. This can be done when the file is opened or at some later time.

Synchronous Writes

The slide shows the data flow of a synchronous write, from the time the `write()` system call is issued, to when the write call returns to the process.

1. The process issues a synchronous `write()` system call.
2. Assuming the process is writing to a new file data block, a new file system block is allocated on disk and an image of that block is allocated in the buffer cache.
3. Once the data is copied from the data area of the process to the buffer cache, an I/O request is placed in the disk I/O queue for that particular disk. The calling process goes to sleep until the write is reported to be complete.

4. When the physical write is performed, the data is first copied from the buffer cache to the firmware cache on the disk drive controller.

NOTE	Most SCSI disk drive controllers can be configured to return an I/O complete acknowledgment at this point, rather than waiting for the data to be transferred to the physical platters. This condition is called "immediate reporting".
-------------	---

5. The data is transferred from the disk controller cache to the platter. This operation is often the most time consuming part of the write, as it involves seek, latency, and data transfer operations.
6. Once the data has been successfully transferred to the platters, the disk drive controller returns an I/O complete acknowledgment to the kernel (assuming this was not done in step 4 with immediate reporting).
7. The kernel, upon receiving the I/O complete acknowledgment, Wakes the sleeping process, which then returns from the write call.

Asynchronous Writes

An asynchronous write does not wait for the data to get to the disk. An asynchronous write system call returns immediately upon the data being written to the buffer cache. In the diagram on the slide, the write call would return following step 2.

The advantage of asynchronous writes is performance — the process does not have to wait for the physical I/O. The disadvantage is lack of data integrity. Because the process continues executing before the data is written to disk, it can perform additional actions that are dependent upon the data being written successfully. If for some reason the data does not get written (a disk goes offline or a disk head crashes), the additional actions can leave the system in an inconsistent state.

For example, assume a database record is written asynchronously. Because it is written asynchronously, the database process continues its execution. A subsequent action is to update a corresponding entry in another table of the database located on another disk. Assume the first asynchronous write is posted to a busy disk with a long queue, and the subsequent write is posted to a disk with an empty queue. The second write finishes before the first write begins! If the system were to crash after the second write, but before the first write, the database would be out-of-sync and corrupted, because the second write assumed that the first write succeeded. There is no "signaling" to the writing process to let it know that a write has completed. For that, the process would have to do synchronous writes.

8-4. SLIDE: Disk Metrics to Monitor — Systemwide

Disk Metrics to Monitor — Systemwide



- Utilization of disk drives
- Disk I/O queue length
- Amount of physical I/O to
 - Device (i.e., Disk)
 - Logical volume
 - File system
- Buffer cache hit ratio

Student Notes

When monitoring disk I/O activity, the main metrics to monitor are:

- **Percent utilization of the disk drives:** As utilization of the disk drives increases, so does the amount of time it takes to perform an I/O. According to the performance queuing theory, it takes twice as long to perform an I/O when the disk is 50% busy, than it does when the disk is idle. Therefore, we consider that a disk may be experiencing a bottleneck if the disk is 50% busy or more.
- **Requests in the disk I/O queue:** The number of requests in the disk I/O queue is one of the best indicators of a disk performance problem. If the average number of requests is above two, then requests are forced to wait in the queue longer than the amount of time needed to service their own requests. If the average number of requests is three or greater, you should also see that the average wait time for a request is greater than the average service time.
- **Amount of physical I/O:** If the amount of disk activity is high, it is important to investigate on which disk, which logical volume, and which file system the activity is occurring on.

- **Buffer cache hit ratio:** One reason disk activity could be high is that read or write requests are not finding corresponding disk blocks in the buffer cache. As a result, physical I/O requests are being generated to the disk.

The read cache hit ratio on the buffer cache indicates how frequently read data is found in the buffer cache. The minimum read hit ratio should be 90% or higher for optimal performance. Less than 90% indicates the buffer cache may be too small, causing (potential) excess disk activity. It may also indicate that the application is not using buffer cache in an efficient manner, e.g. doing a lot of random I/O or very large I/O.

The write cache hit ratio on the buffer cache indicates how frequently a write to a buffer does not trigger a physical read or write to the disk. (if only a portion of a block is being written, and the image of that block is not already in a buffer, it may be necessary to read the original contents of the block into buffer cache before modifying it with the new write data.) The minimum write cache hit ratio should be 70% or higher for optimal performance. Less than 70% indicates the buffer cache may be too small, causing (potential) excess disk activity. Again, the fault may lie with the application's use of the buffer cache.

8-5. SLIDE: Disk Metrics to Monitor — Per Process

Disk Metrics to Monitor — Per Process



- Amount of physical and logical I/O being performed on a per process basis
- Type and amount of system calls (I/O-related) being generated by processes performing large amounts of I/O
- Paging to swap device (VM read/writes) on a per process basis
- Files opened by processes performing large amounts of I/O

Student Notes

On a per process basis, it is important to identify which processes are generating large amounts of disk I/O. Metrics that help to identify I/O activity on a per process basis are:

- **Amount of physical and logical I/O:** This indicates “how much” I/O the process is performing. For processes performing large amounts of I/O, the additional three metrics shown below should be investigated.
- **Type and amount of I/O related system calls being generated:** For each process performing high I/O, the number of `read()`, `write()`, and other I/O related calls should be inspected.
- **Amount of VM reads and VM writes:** If the I/O activity being generated is due to paging (VM read and VM writes), then the problem is probably not a disk I/O problem, but more like a memory problem.
- **Files opened with heavy access:** For each process performing large amounts of file system I/O, the names of the files to which they are reading or writing should be inspected. For files receiving high I/O activity, consider relocating these files to other disks that are less busy. To determine how “random” the I/O requests are, hit <CR>

frequently while looking at the list of open files for that process (in **g**lance), then inspect how quickly the offset to each file changes and whether it is monotonically increasing or varies up and down.

8-6. SLIDE: Activities that Create a Large Amount of Disk I/O

Activities that Create a Large Amount of Disk I/O



- Buffer cache misses
- Synchronous I/O
- Accessing sequentially with a small block size
- Accessing many files on a single disk
- Accessing many disk drives from a single disk controller card

Student Notes

Common causes of disk-related performance problems are shown on the slide.

- Buffer cache misses cause physical I/O to occur. When the appropriate buffer is not found in the buffer cache, a physical I/O is triggered. By the way, a buffer cache can be too large as well. A very large buffer cache takes more time to search to see if the appropriate buffer exists! More on how to properly size a buffer cache will be given later in this module.
- Synchronous I/O forces the write system calls to wait until the I/O physically completes. Very good for data integrity, very poor for performance.
- Sequential access, with a small block size, causes excessive amounts of physical I/O.
- Accessing lots of files on one disk, versus many disks, creates an imbalance of disk drive utilization. This leads to performance problems with the busy disks and under utilization with the less busy disks.
- Accessing lots of disks on the same disk controller creates contention problems on the SCSI bus. You can determine this by noticing that multiple disks on the same controller

have request queues that are consistently three or greater in length and the average time a request waits to be serviced is greater than the average time it takes to actually service the request. The individual disks may not show a disk utilization 50% or greater!

If this situation occurs, it would be best to spilt up the busiest disks onto separate controllers.

8-7. SLIDE: Disk I/O Monitoring sar -d Output

Disk I/O Monitoring sar -d Output							
# sar -d 5 6							
05:23:50	device	%busy	avque	r+w/s	blks/s	await	avserv
05:23:55	c1t5d0	0.60	0.50	2	35	1.55	1.07
	c0t4d0	62.40	10.51	46	2783	127.97	152.92
	c0t5d0	33.20	2.76	16	1226	42.89	141.96
	c0t6d0	54.80	8.10	31	2166	242.52	193.15
05:24:00	c1t5d0	1.20	0.50	3	39	1.97	0.72
	c0t4d0	63.80	10.84	48	2943	129.23	159.47
	c0t5d0	39.20	2.94	19	1427	38.85	151.55
	c0t6d0	61.80	19.60	36	2371	331.15	208.49
05:24:05	c1t5d0	2.20	0.50	3	45	3.85	1.04
	c0t4d0	56.40	18.40	39	2392	234.33	161.10
	c0t5d0	35.60	2.69	17	1258	39.96	138.81
	c0t6d0	62.80	18.41	36	2643	192.28	170.66
05:24:10	c1t5d0	0.20	0.50	2	35	1.01	0.86
	c0t4d0	68.60	13.00	51	3118	154.68	159.02
	c0t5d0	33.80	3.25	16	1226	47.82	147.32
	c0t6d0	60.00	5.72	33	2301	238.43	208.88
05:24:15	c0t4d0	24.40	4.25	15	823	60.83	180.68
	c0t5d0	23.00	3.46	14	851	43.33	118.87
	c0t6d0	50.60	18.77	28	1846	306.13	238.36
05:24:20	c1t6d0	0.60	0.50	0	2	4.63	11.53
	c1t5d0	1.40	1.17	2	23	9.85	21.50

Student Notes

The `sar -d` report shows disk activity on a per disk drive (spindle) basis. The key fields within this report are:

- % busy** Indicates the average percent utilization of the disk over the interval (5 seconds in the slide).
- avque** Indicates the average number of requests in the disk I/O queue.
- await** Indicates the average amount of time a requests spends waiting in the disk I/O queue.
- avserv** Indicates the average amount of time to service a disk I/O request.

The `sar -d` report on the slide shows that when the disk had the most requests in the queue (19.60 and 18.77), the average wait time was at its highest.

The slide also shows that there are five disk drives spread across two disk controllers. One disk controller (c0) appears to have two busy drives (t4 and t6), and a relatively low usage drive (t5). Disk controller (c1) has two disks that are mainly idle. One performance solution

here would be to balance the disk activity across the two controllers by moving one disk (say c0t4) over to the less busy disk controller (c1).

8-8. SLIDE: Disk I/O Monitoring sar -b Output

Disk I/O Monitoring sar -b Output									
buffer code l'info est fournie au niveau de buffer cache									
#=> sar -b 10 20									
HP-UX e2403rec B.10.20 U 9000/856 02/09/98									
	bread/s	lread/s	%rcache	bwrit/s	lwrit/s	%wcache	pread/s	pwrit/s	
05:51:04	0	0	0	1	1	25	0	0	
05:52:04	0	0	0	0	1	85	0	0	
05:52:14	0	0	0	1	8	87	0	0	
05:52:24	0	0	0	0	4	100	0	0	
05:52:34	0	0	0	0	1	100	0	0	
05:52:54	1	68	99	0	0	33	0	0	
05:53:04	7	11936	100	1	2	13	0	0	
05:53:14	6	19506	100	1	1	0	0	0	
05:53:24	28	24147	100	1	2	65	0	0	
05:53:34	64	16659	100	0	14	99	0	0	
05:53:44	118	118	0	2	3	46	0	0	
05:53:54	0	0	0	3	3	0	0	0	
05:54:04	0	0	0	18	19	4	0	0	
05:54:14	179	179	0	18	18	3	0	0	
05:54:24	179	179	0	13	14	4	0	0	
Average	29	3639	99	3	5	39	0	0	
continue ds buffer cache									

Student Notes

The `sar -b` report shows disk activity related to the buffer cache. The key fields within this report are:

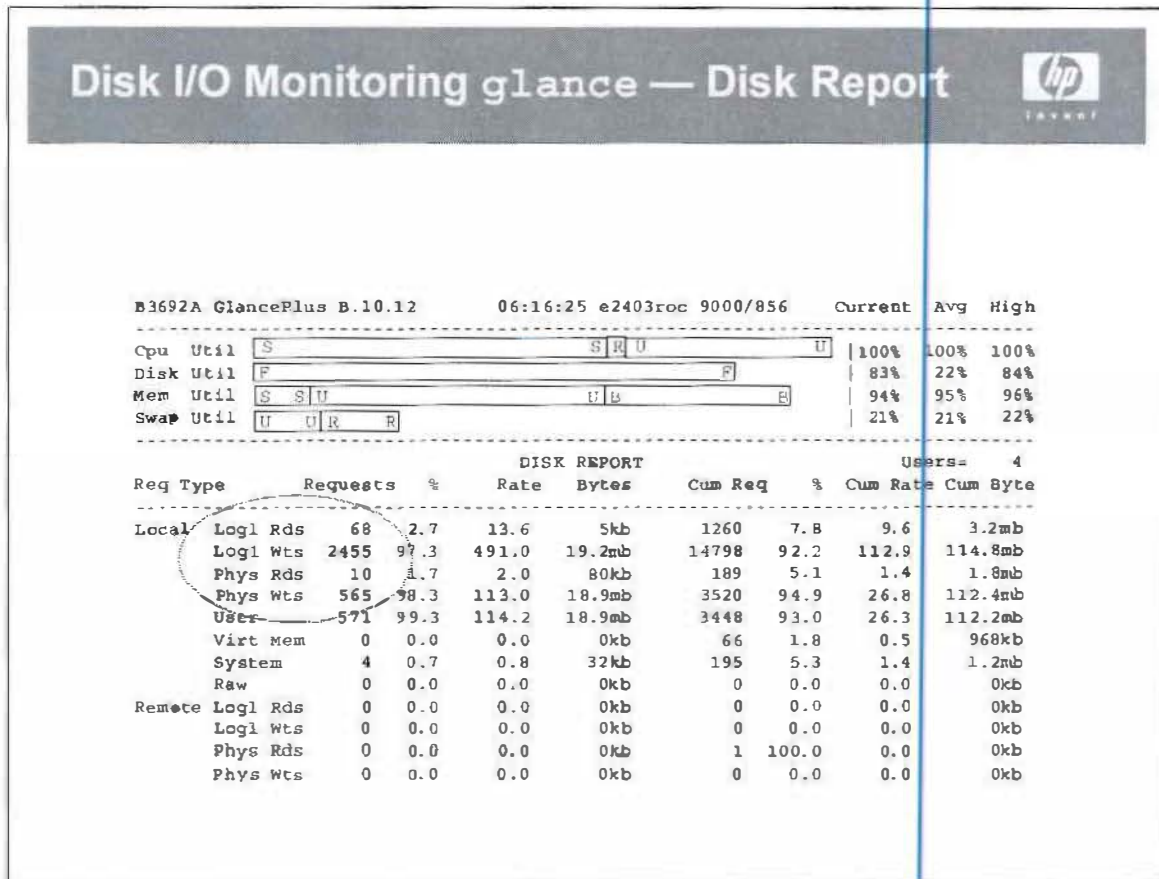
bread/s	Indicates the average number of physical I/O reads per second over the interval. The term bread refers to block reads.
lread/s	Indicates the average number of logical I/O reads per second over the interval.
%rcache	Indicates the average percent read cache hit rate. This shows what percentage of read requests were satisfied through the buffer cache. Ideally, this value should be consistently 90% or greater.
bwrit/s	Indicates the average number of physical I/O writes per second over the interval. The term bwrit refers to block writes.
lwrit/s	Indicates the average number of logical I/O writes per second over the interval.

%wcache Indicates the average percent write cache hit rate. This shows what percentage of write requests were satisfied through the buffer cache. Ideally, this value should be consistently 70% or greater.

The **sar -b** report on the slide shows the two extreme situations. The first extreme is a 100% cache hit rate, which occurs when there are lots of logical I/O requests and all requests are satisfied through the buffer cache, rather than having to go to disk. This is a very desirable condition.

The other extreme is a 0% cache hit ratio. This occurs when every logical I/O request required a physical I/O from disk. In this case, the number of physical reads or writes is equal to the number of logical reads or writes. This is most undesirable.

8-9. SLIDE: Disk I/O Monitoring glance — Disk Report



Student Notes

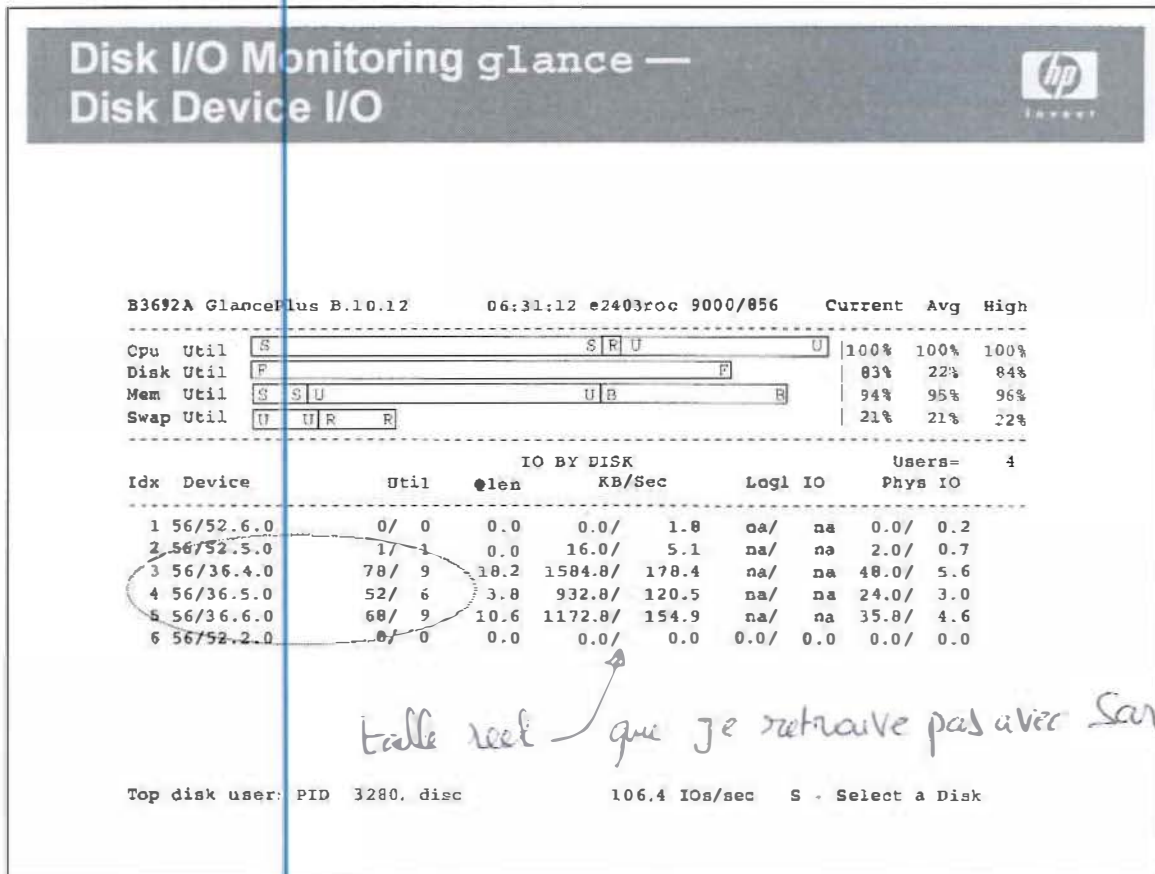
The **glance** disk report (**d** key) shows local and remote I/O activity. The I/O distribution can be viewed from the following:

- Logical Perspective (logical reads and logical writes)
- Physical Perspective (physical reads and physical writes)
- I/O Type Perspective (User, Virtual Mem, System, Raw)

Items of interest in this report include the number of logical I/O requests (read and writes), the number of physical I/O requests (reads and writes), and the ratio between the two.

In the slide, disk utilization is 94% (very high), with the majority of the I/Os being writes (92%) as opposed to reads. It is also interesting to note the logical to physical write ratio is 14,798 / 3,520 or approximately 4:1, which is an acceptable write performance ratio.

8-10. SLIDE: Disk I/O Monitoring glance — Disk Device I/O



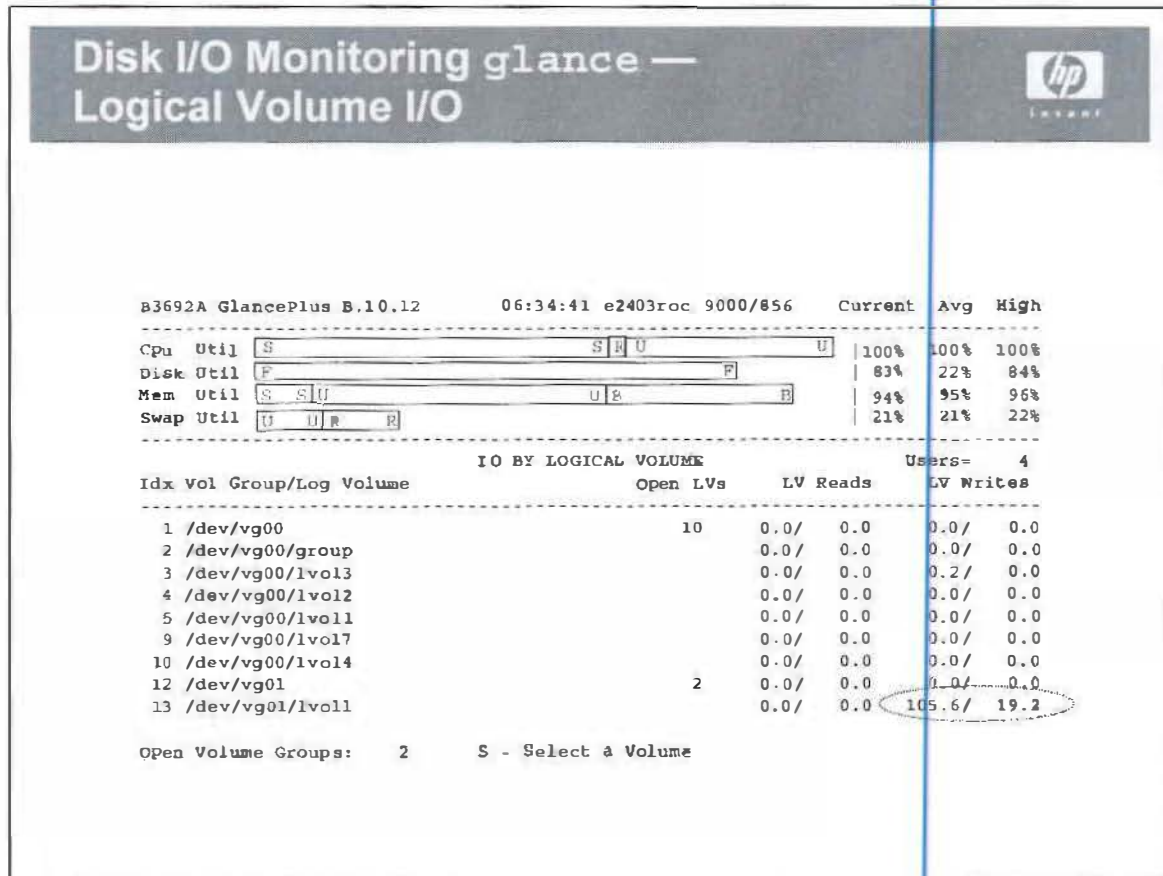
Student Notes

The **glance** disk device report (u key) shows current and average utilization of each disk drive on the system. The report also shows the current I/O queue length for each disk. This display shows basically the same information as **sar -d**.

In the slide, three disks show utilization greater than 50% and queue lengths greater than 3. This is normally a valid reason for further investigation. The 10.6 and 18.2 queue lengths are high, but, because the average utilization of both the drives is 9%, this may just be a spike in disk activity.

In this case, monitor the situation further to see if the high queue lengths persist or if they were just spikes in disk usage.

8-11. SLIDE: Disk I/O Monitoring glance — Logical Volume I/O

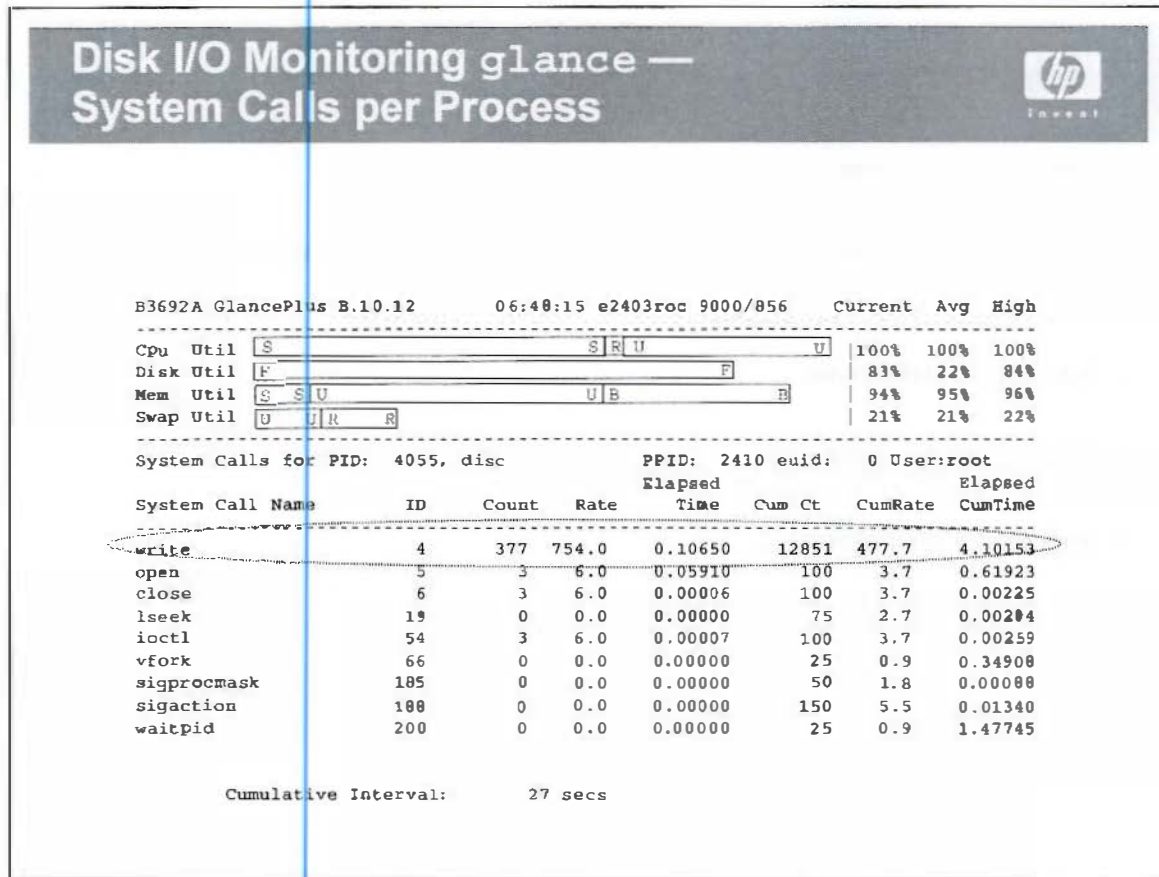


Student Notes

The **g**lance logical v volume report (**v** key) shows disk activity on a per logical volume basis. Only physical I/O activity (not logical I/O activity) is shown with this report.

In the previous slide, we saw high activity across three disk drives (drives 4, 5, and 6). The logical volume report on the slide shows all this activity is being performed against one logical volume (**/dev/vg01/lvol1**), which implies that the logical volume is being spread across three disks (a good idea since the I/O to the logical volume is so high).

8-12. SLIDE: Disk I/O Monitoring glance — System Calls per Process



Student Notes

The glance system calls report (L key), available only from the select process report (s key), shows the names of the system calls being generated by the selected process. The system calls report can be viewed for individual processes (as shown on the slide), or globally for all processes on the system (Y key).

Significant system calls, which typically consume a lot of time, are the file I/O related calls, such as `read()`, `write()`, `open()`, and `close()`.

In the slide, the `write()` system call is being invoked heavily by the selected process (754 times/second) and has accounted for 4.1 seconds of the CPU's time over a 27-second period (approximately 15%).

8-13. SLIDE: Tuning a Disk I/O-Bound System — Hardware Solutions

Tuning a Disk I/O-Bound System — Hardware Solutions



- Add additional disk drives (and off load busy drives).
- Add additional controller cards (and balance disk drive load across controllers).
- Add faster disk drives.
- Implement disk striping.
- Implement disk mirroring.

Student Notes

The hardware solutions on the above slide will help to lessen the performance impact of high disk I/O on a system.

- **Add more disk drives and load balance across disks.** This spreads the amount of I/O over more drives, decreasing the average number of I/O requests for each disk. Many smaller disks are better than a few large disks.
- **Add more disk controllers and balance load across disk controllers.** This spreads the amount of I/O over more controllers, decreasing the likelihood that any one disk controller will become overloaded with I/O requests.
- **Add faster disk drives.** This decreases the amount of time it takes to service an I/O request, which decreases the amount of time requests spend waiting in the disk I/O queue.
- **Implement disk striping.** This increases the number of disk heads having access to the striped data (the more disks striped across, the more heads accessing the data,

simultaneously). It also allows for “overlapping seeks,” meaning that one disk head can be seeking the next block, while a second disk head is reading the current data block.

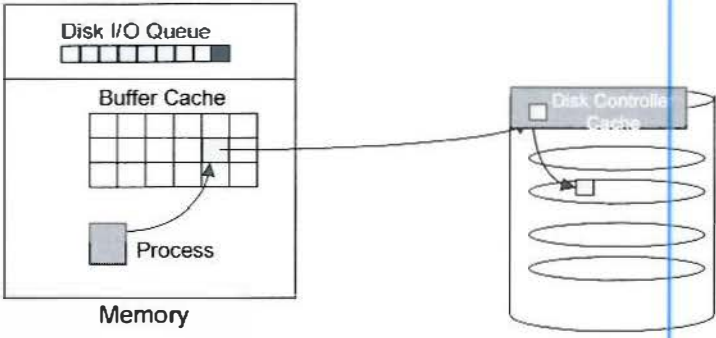
- **Implement disk mirroring.** This can increase read performance, as either the primary or mirrored copy of the data can be read. In fact, the data will be read from whichever disk has the fewest I/Os pending against it. However, it will negatively impact write performance. In order to maintain the integrity of the mirrors, duplicate writes must be done to each copy of the mirrored volume/disk. Mirroring is primary a data integrity feature, but under the right circumstances (read-intensive data) it can improve performance, as well.

8-14. SLIDE: Tuning a Disk I/O-Bound System — Perform Asynchronous Meta-data I/O

Tuning a Disk I/O-Bound System — Perform Asynchronous I/O



- Configure individual disk drives to behave “somewhat” asynchronously with immediate reporting feature of SCSI disk controllers.
- Configure immediate reporting with the `scsictl` command.



Student Notes

Asynchronous I/O significantly improves write performance over synchronous I/O because the write requests (and thus the requesting processes) do not have to wait for the data to be written to the disk platters.

Immediate Reporting for Selected Disks

Immediate reporting can be turned on at boot time by setting the tunable parameter `default_disk_ir` to “ON”. An alternative to turning on `default_disk_ir` is to enable certain disk controllers selectively to report immediately to the kernel when the data reaches the disk controller cache. For normal writes, the disk waits until data is transferred from the controller cache to the disk platters, before returning to the kernel. By setting `immediate reporting` to ON for individual disk controllers, processes do not have to wait for the seek or latency times when writing to those disks.

The `scsictl` command can be used to turn immediate reporting ON (1) for a particular SCSI disk. The default for immediate reporting is OFF (0).

Examples

To view the device settings for the controller at SCSI adapter address "0" and SCSI target address 6:

```
# /usr/sbin/scsictl -m ir /dev/rdisk/c0t6d0  
immediate_report = 0
```

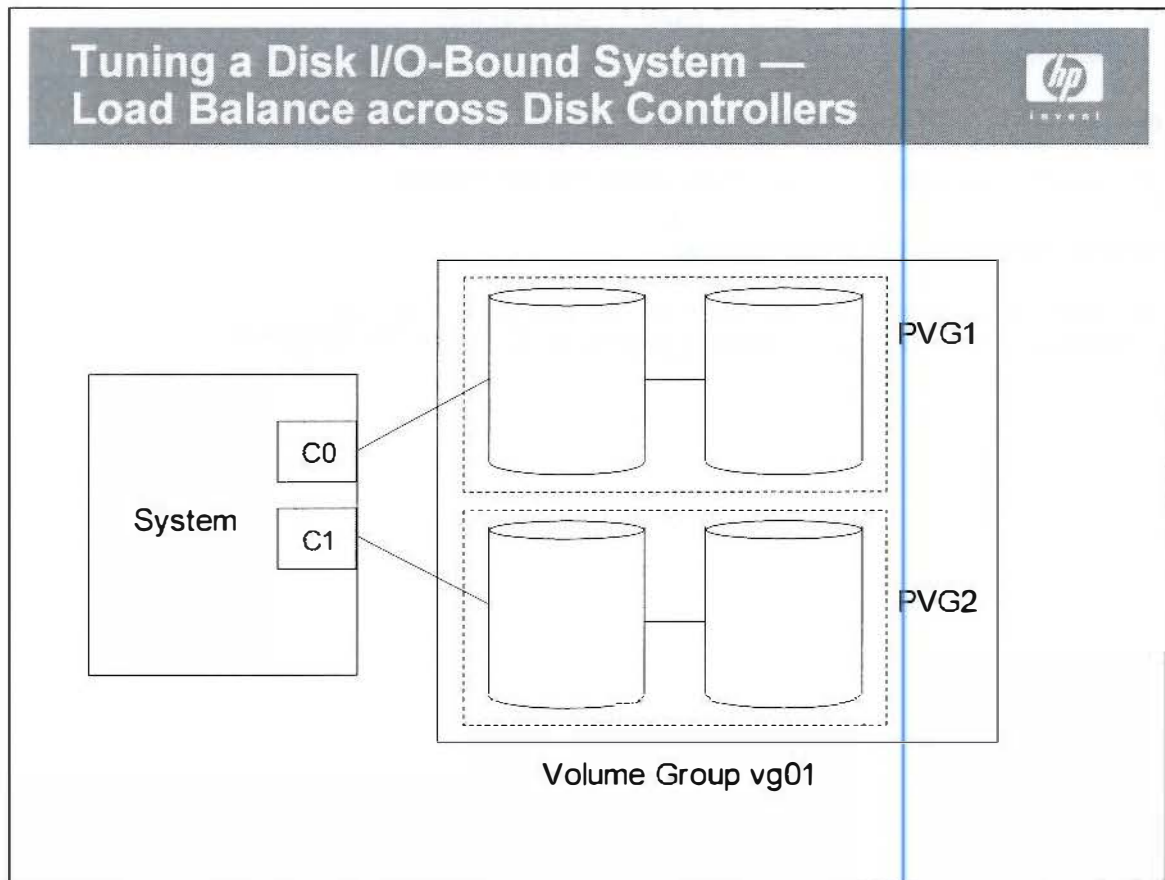
To change the value of immediate reporting to ON:

```
# /usr/sbin/scsictl -m ir=1 /dev/rdisk/c0t6d0
```

To view the changes in the device settings:

```
# /usr/sbin/scsictl -a /dev/rdisk/c0t6d0  
immediate_report = 1; queue_depth = 8 limita la queue
```

8-15. SLIDE: Tuning a Disk I/O-Bound System — Load Balance across Disk Controllers



Student Notes

Another potential solution to a disk I/O performance problem is to spread the write requests across the disk controllers as evenly as possible. This helps ensure no one controller becomes overloaded with I/O requests.

Mirroring Logical Volumes

A popular feature of LVM is the ability to mirror logical volumes to separate disk drives. This involves writing one copy of the data to the primary disk and one copy to the mirrored disk. When the primary disk and mirror disk are on the same disk controller, a performance bottleneck often results because the disk controller has to service the writes for both the primary and mirrored data.

Physical Volume Groups

Physical volume groups (PVGs) allow disk drives to be grouped, based on the disk controller to which they're attached. Used in conjunction with LVM mirroring, it ensures the mirrored data not only goes to a different disk, but also goes to a different PVG group (that is, a different disk controller).

How to Set Up PVGs

The PVG groups are defined in the `/etc/lvmmpvg` file. This file can be manually edited or updated with the `-g` option to the `vgcreate` and `vgextend` commands.

A sample `/etc/lvmmpvg` file, based on the four disks on the slide are:

```
VG /dev/vg01
PVG      PV_group0
    /dev/dsk/c0t6d0
    /dev/dsk/c0t5d0
PVG      PV_group1
    /dev/dsk/c2t5d0
    /dev/dsk/c2t4d0
```

Configuring LVM to Mirror to Different PVGs

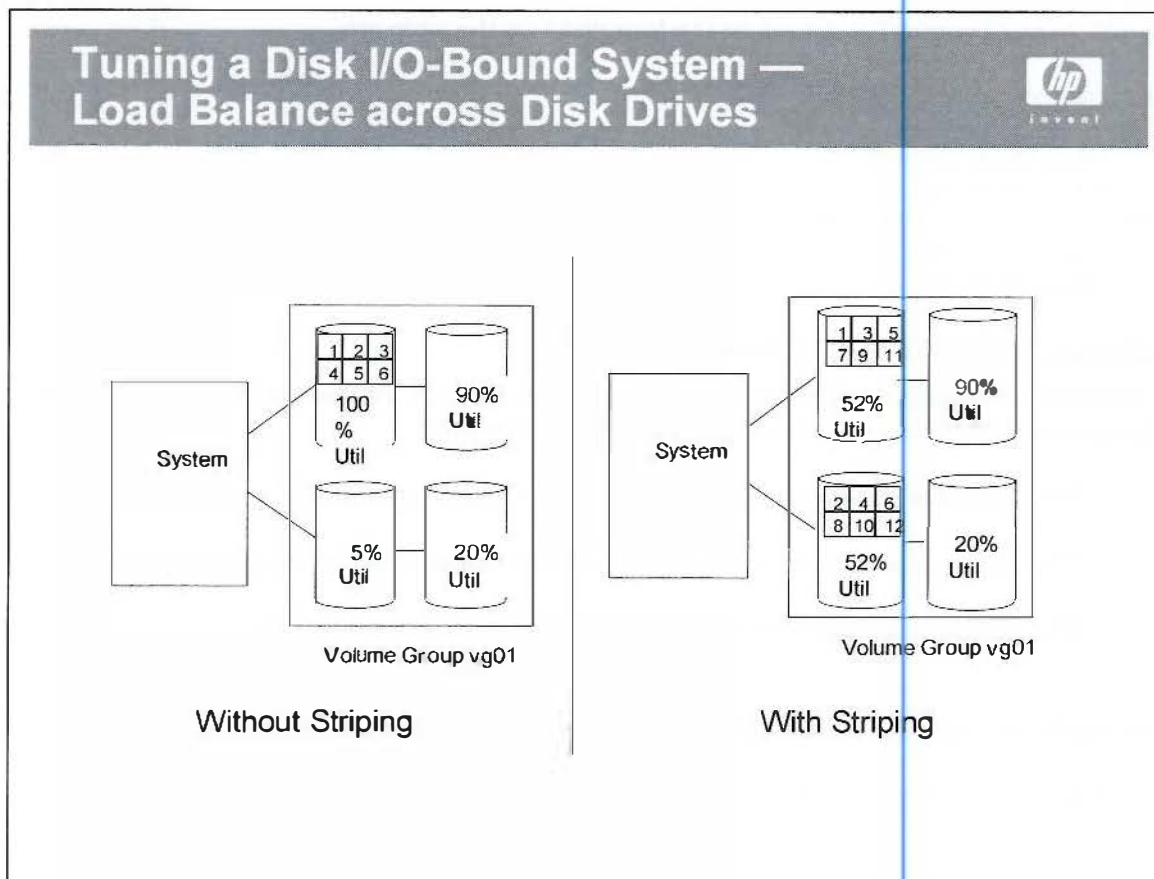
The command to configure LVM mirroring for different PVGs is `lvchange`. The strict option to this command, `-s`, contains the following three arguments:

- y** This indicates all mirrored copies must reside on different disks.
- n** This indicates mirrored copies can reside on the same disk as the primary copy.
- g** This indicates all mirrored copies must reside with different PVGs.

For example, to configure `/dev/vg01/lvol1` to mirror to different PVG:

```
lvchange -s g /dev/vg01/lvol1
```

8-16. SLIDE: Tuning a Disk I/O-Bound System — Load Balance across Disk Drives



Student Notes

Balancing the disk activity so that the utilization across drives is approximately the same helps to ensure that no one disk becomes overloaded with I/O requests (that is, 50% or greater utilization, with three or more requests in the disk queue).

The slide illustrates a situation in which one disk is heavily utilized (100%) while another disk is only 5% utilized. One potential solution is to stripe the heavily utilized logical volume on the first disk to both disks.

LVM Striping

The ability to stripe a logical volume across multiple disks (at a file system block level) was introduced into LVM at the HP-UX 10.01 release. A logical volume must be configured for striping at the time of creation. Once a logical volume is created, it cannot be striped without recreating the logical volume.

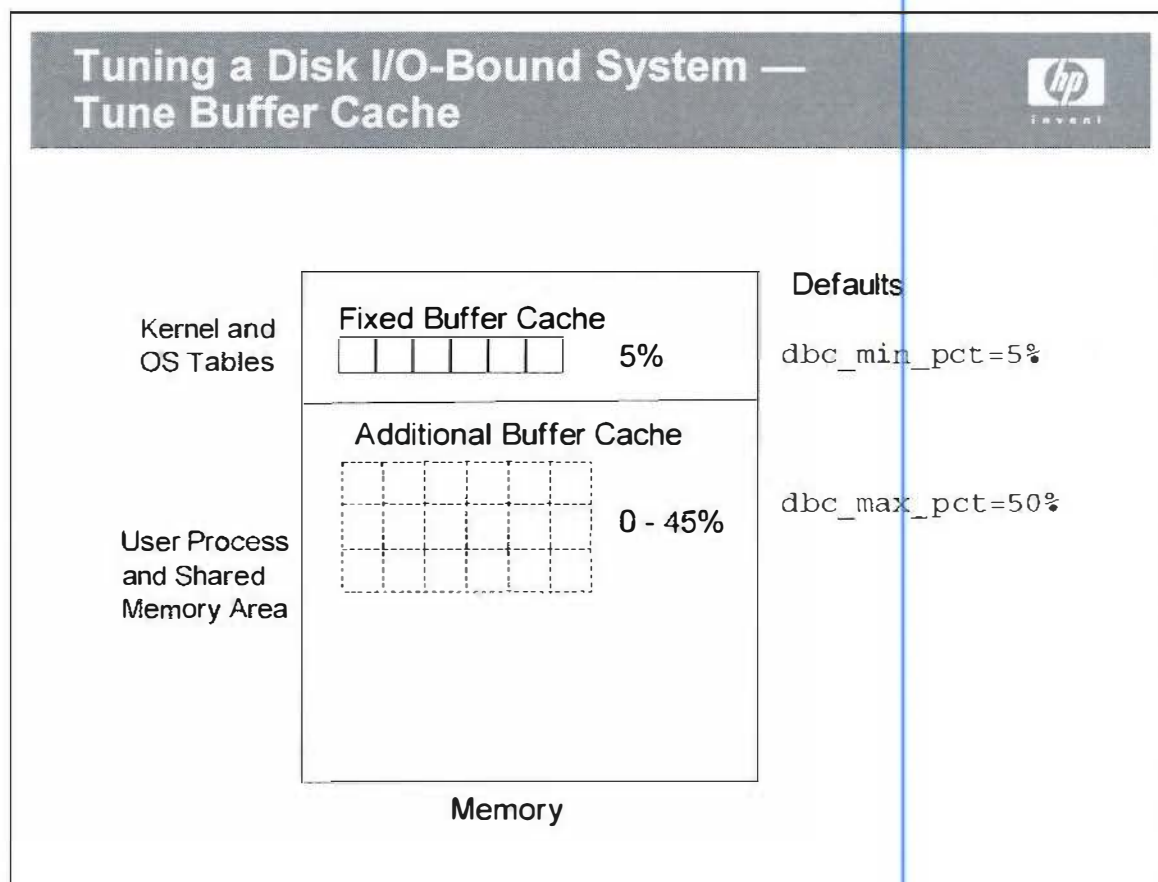
The command to create a striped logical volume is `lvcreate`. The syntax, related to striping, for this command is:

```
lvcreate -i [number of disks] -I [stripe size] -L [size in MB] vg_name
```

Example:

```
lvcreate -i 2 -I 8 /dev/vg01  
lvextend -L 50 /dev/vg01/lvol2 /dev/dsk/c0t5d0 /dev/dsk/c0t4d0
```

8-17. SLIDE: Tuning a Disk I/O-Bound System — Tune Buffer Cache



Student Notes

With the introduction of HP-UX 10.0, the buffer cache becomes dynamic, growing and shrinking between a minimum size and a maximum size.

NOTE: Space for the buffer cache is allocated in two different areas of memory: the minimum size is created in the O/S area of memory, and anything above the minimum size is allocated from the User Process area.

How the Buffer Cache Grows

As the kernel reads in files from the file system, it will try to store the data in the buffer cache. If memory is available and the buffer cache has not reached its maximum size, the kernel will grow the buffer cache to make room for the new data. As long as there is memory available, the kernel will keep growing the buffer cache until it reaches its maximum size (50% of memory, by default).

If memory is not available, or the buffer cache is at its maximum size when new data is read, the kernel will select buffer cache entries that are least likely to be needed in the future, and reallocate those entries to store the new data.

The main point is that if there is available memory, the buffer cache will grow into this memory until there is no memory left (or until the buffer cache reaches its maximum size).

How the Buffer Cache Shrinks

As memory falls below `LOTSFREE`, the `vhand`-paging daemon wakes up and begins paging out 4-KB pages of memory. The eligible pages include process segments (text, data, and stack), shared memory segments, and the buffer cache. In other words, the buffer cache is shrunk by having `vhand` page out its pages. The buffer cache is treated by `vhand` as just another structure in memory with pages that it can *dereference* and free. Like process text pages, buffer cache pages are not written out to the swap space. But, since their contents may have been modified, they could be flushed out to the file system being placed back on the free page list.

NOTE: It should be noted that the kernel global value, `dbc_steal_factor`, determines how aggressive the `vhand` daemon is at stealing buffer cache pages in comparison to process pages. A value of 16 says to treat buffer cache pages no differently than process pages; the default value of 48 says to steal buffer cache pages three times as aggressively! However, if the buffer cache is referencing those pages, `vhand` will find few buffers to free up.

Buffer Cache Performance Implications

Because the buffer cache grows quickly into free memory and shrinks slowly by necessitating `vhand` to page it out, one consideration is to limit the maximum size to which the buffer cache can grow.

The default maximum size is 50% of total memory. This probably was a fairly reasonable number when the parameter was introduced, but with the very large memory systems existing nowadays, it's probably much too high. By setting the `dbc_max_pct` tunable kernel parameter to a smaller number (say 20 or 25), the buffer cache can still grow to a significant size, but will not be so large that it takes a long time to shrink when more processes become ready to execute.

Prior to HP-UX11i, there was a definite performance penalty for having a buffer cache that was too large. It took a long time to search the cache to determine if the needed buffer was already there. Improvements in the search algorithm in 11i have reduced that penalty significantly.

Fixed vs. Dynamic Buffer Cache

Should you use a fixed-size buffer cache or a dynamic buffer cache? If your buffer cache requirements are constant over time, of course you should use a fixed-size buffer cache. Simply set the `dbc_min_pct` and `dbc_max_pct` parameters to the same value.

If your buffer cache requirements change over time, do they change rapidly or slowly. There is some overhead associated with growing and shrinking buffer cache. Plus, shrinking buffer cache is not a very fast operation. If your buffer cache changes slowly over time, it would be best to use a dynamic buffer cache. The overhead of growing and shrinking would be spread out and become relatively insignificant.

Module 8

Disk Performance Issues

If, however, your buffer cache requirements change rapidly over time, you probably would be better served with a fixed-size buffer cache, properly sized to give you adequate buffers most of the time. Only on relatively rare occasions, would buffer cache be a bottleneck and only for short periods. In the long run, your performance would be better than trying to deal with the rapidly changing needs using a dynamic buffer cache.

Sizing Buffer Cache

Here is a set of recommendations for properly sizing your buffer cache.

1. Are you getting at least a 90% read cache hit rate and a 70% write cache hit rate? If so, your buffer cache may already be larger than necessary. If you are experiencing no memory pressure, and no apparent disk bottlenecks, leave the buffer cache as it is.
2. If you are experiencing memory pressure or apparent disk bottlenecks, try slinking the size of your buffer cache. Adjust `dbc_max_pct` down, in increments, no more than 10% at a time, until your performance figures fall to 90%/70%.
3. If you are not getting 90%/70% performance from your buffer cache, it may be too small or your application may be using it in an inefficient manner. Try increasing its size. If the figures improve, keep increasing the size until either you reach 90%/70% or your performance ceases to improve. Leave the size there.
4. If increasing the size of the buffer cache does not produce an immediate improvement in performance, your application may need to be tuned to use the buffer cache more efficiently. However, your buffer cache may still be larger than it needs to be. After you have tuned your application, recheck your buffer cache performance, as above.

8-18. LAB: Disk Performance Issues

Directions

The following lab illustrates a number of performance issues related to disks.

1. A file system is required for this lab. One was created in an earlier exercise. Mount it now.

```
# mount /dev/vg00/vxfs /vxfs
```

We also need to assure that the controller does not have "SCSI immediate reporting" enabled. Enter the following command and check your current state: (fill in the device file name as appropriate)

```
# scsictl -m ir /dev/rdisk/cXtXdX (to report current "ir" status)
```

If the current `immediate_report = 1` then enter the following:

```
# scsictl -m ir=0 /dev/rdisk/cXtXdX (ir=1 to set, ir=0 to clear)
```

2. Copy the lab files to the file system.

```
# cp /home/h4262/disk/lab1/disk_long /vxfs
```

```
# cp /home/h4262/disk/lab1/make_files /vxfs
```

Next, execute the `make_files` program to create five 4-MB ASCII files.

```
# cd /vxfs
```

```
# ./make_files
```

3. Purge the buffer cache of this data, by unmounting and remounting the file system.

```
# cd /
```

```
# umount /vxfs
```

```
# mount /dev/vg00/vxfs /vxfs
```

```
# cd /vxfs
```

Module 8
Disk Performance Issues

4. Open a second terminal window and start **glance**. While in **glance**, display the Disk Report (**d** key). Zero out the data with the **z** key.

From the first window, time how long it takes to read the files with the **cat** command. Record the results below:

```
# timex cat file* > /dev/null      glance Disk Report
```

real:	
user:	Logl Rds:
sys:	Phys Rds:

5. At this point, all 20 MB of data is resident in the buffer cache. Re-execute the same command and record the results below:

```
# timex cat file* > /dev/null      glance Disk Report
```

real:	
user:	Logl Rds:
sys:	Phys Rds:

NOTE: The conclusion is that I/O is much faster coming from the buffer cache, than having to go to disk to get the data.

6. The **sar -d** report.

Exit **glance**, and in the second window start: **# sar -d 5 200**

From the first window, execute the **disk_long** program, which writes 400 MB to the HFS file system (and then removes the files).

```
# timex ./disk_long
```

- How busy did the disk get?
- What was the average number of request in the I/O queue?
- What was the average wait time in the I/O queue?
- How much real time did the task take?

7. The **glance I/O by Disk** report

Exit from the **sar -d** report, and start **glance** again. While in **glance**, display the I/O by Disk report (**u** key). From the first window, re-execute **disk_long**, timing the execution. Record results below:

```
# ./disk_long      glance I/O by Disk Report
```

Util:	qlen:
-------	-------

8. The **glance** I/O by File System report

Reset the data with the **z** key, and display the I/O by File System report (**i** key).
From the first window, re-execute **disk_long**, timing the execution. Record results below:

```
# ./disk_long
```

glance I/O by Disk Report	
Log I/O:	Phys I/O:

9. Performance tuning — immediate reporting. Ensure the immediate reporting options are set for the disk that the file system is located on. If immediate reporting is not set, set it.

```
# scsictl -m ir /dev/rdisk/cXtXdX (to report current "ir" status)
# scsictl -m ir=1 /dev/rdisk/cXtXdX (ir=1 to set, ir=0 to clear)
```

Purge the contents of buffer cache.

```
# cd /
# umount /vxfs
# mount /dev/vg00/vxfs /vxfs
# cd /vxfs
```

10. The **sar -d** report.

Exit **glance**, and in the second window start: **# sar -d 5 200**

From the first window, execute the **disk_long** program (which writes 400 MB to the file system and then removes the files).

```
# timex ./disk_long
```

- How busy did the disk get?
- What was the average number of requests in the I/O queue?
- What was the average wait time in the I/O queue?
- How much real time did the task take?

How do the results of step 11 compare to the results in step 6?

Module 8
Disk Performance Issues