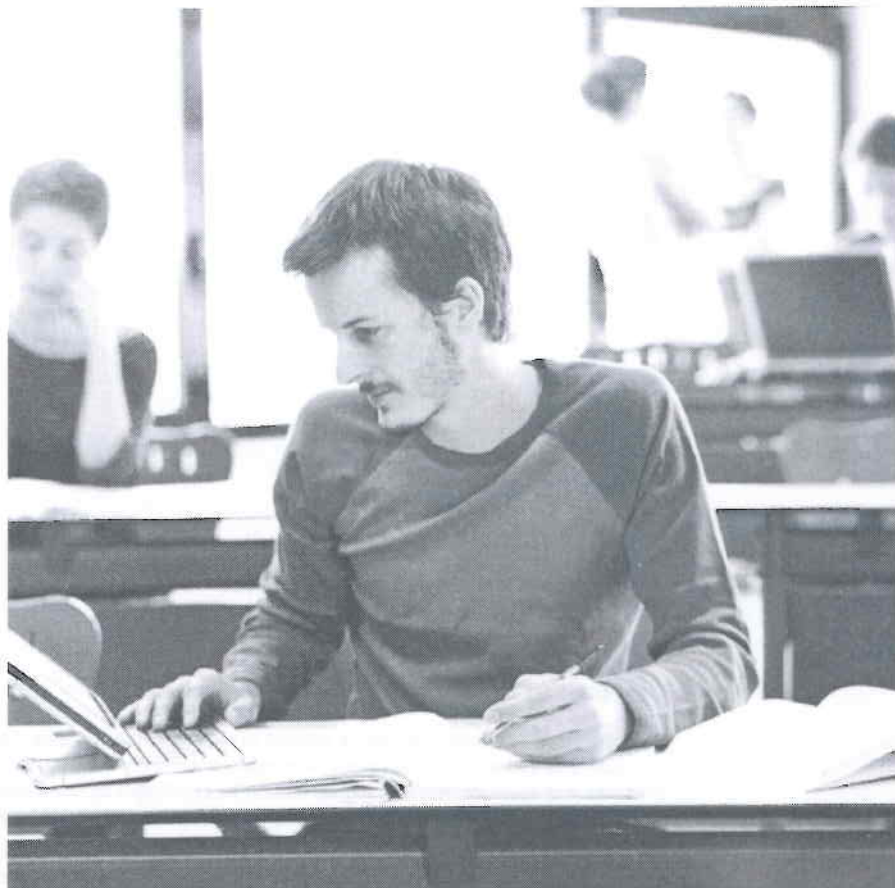




HP-UX Performance and Tuning

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

Module 9 — HFS File System Performance

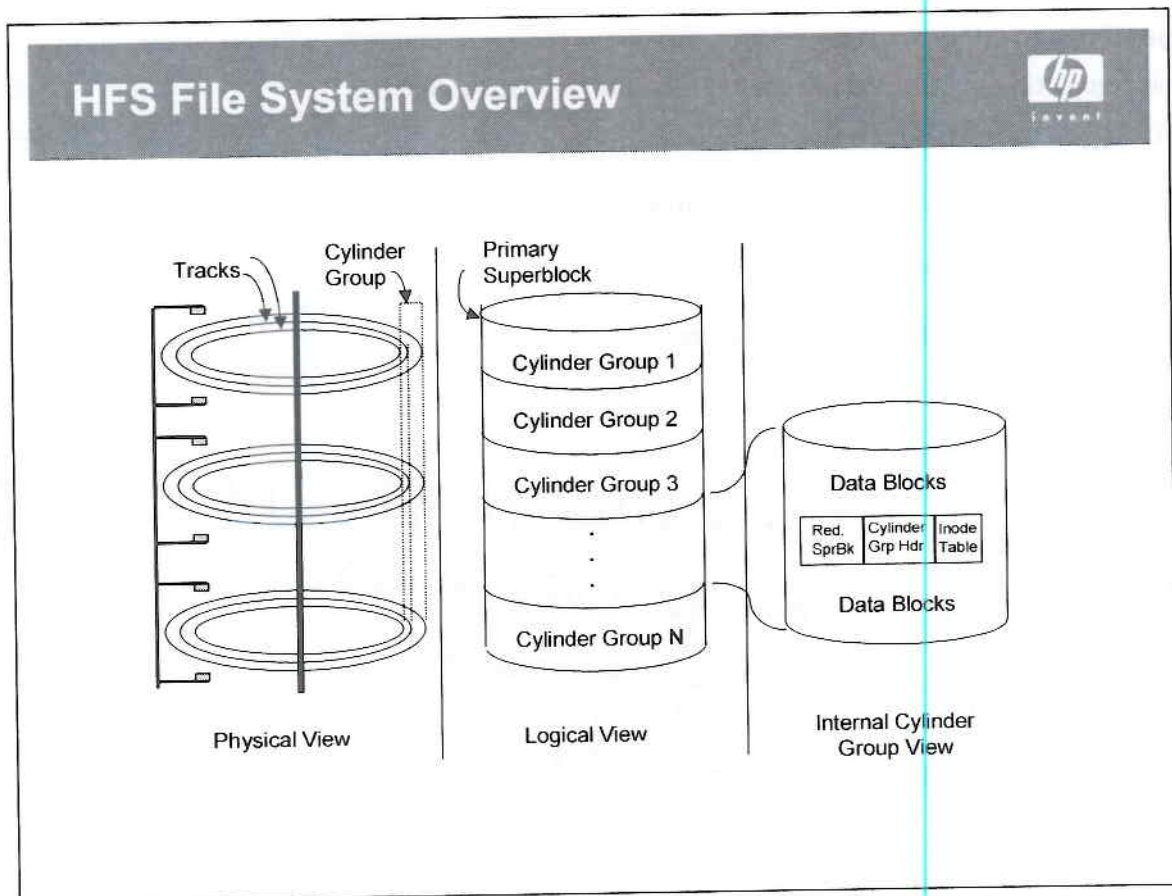
Objectives

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

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

HFS :
VXFS : Veritas External file system (Hp appelle jfs)
CDFS : mntage CDR et DVD
NFS : file system reseau
CIFS : ← samba

9-1. SLIDE: HFS File System Overview



Student Notes

The HFS model is a foundation for all other file system variants. We will begin our discussion of File System performance using the HFS file system model.

The HP-UX File System

The HFS file system strategically lays out its data structures on disk to most efficiently utilize the geometry of the disk. The design of the HFS file system can best be explained by looking at the file system from three perspectives.

Physical View

From a physical disk perspective, the disk drive upon which a file system is placed contains sectors, tracks, platters, and disk heads.

A key behavior of most all disk drives is that the disk heads move in parallel across the platters in such a way that each disk head is over the same track within each platter at the same time. To maximize the file system I/O throughput of the disk, it is desirable to have as many file blocks close to each other as possible, to minimize the time it takes to read or write the various blocks of a file. To help achieve this goal, the blocks on the disk are allocated to

the HFS file system in units call **cylinder groups**. A cylinder group is all the tracks, from every platter, grouped together, of several adjacent cylinders.

Cylinder Group 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 group would be the same group of lanes from each floor's jogging track. In other words, all lane 1, 2, and 3 tracks would make up cylinder group 1; all lane 4, 5, and 6 tracks would make up cylinder group 2, etc.

By organizing space on disks in cylinder group units, the HFS file system can logically keep all the blocks of a given file close to each other. For example, in the slide above, the first 6 blocks of a file might be allocated as follows:

File block #1:	Platter #1, Track #1, Sector #1
File block #2:	Platter #1, Track #1, Sector #2
File block #3:	Platter #1, Track #3, Sector #5
File block #4:	Platter #1, Track #3, Sector #6
File block #5:	Platter #2, Track #7, Sector #10
File block #6:	Platter #3, Track #9, Sector #7

By allocating file system space in this manner, a multiple block read (say 6 blocks) could be read with less than six separate reads. In the example above, file blocks 1 and 2 could be read with one read operation, followed by a head switch (no carriage movement) to track 3, another read for file blocks 3 and 4, a short seek to the next cylinder and a head switch to read file block 5, and repeat for file block 6. Four reads could then read the six blocks. The more contiguous the blocks that make up the file, the more efficient the reads and writes can be.

Logical View

From a logical perspective, an HFS file system contains a series of cylinder groups. Even though the physical cylinder groups are laid out from top to bottom, transcending all the platters, logically, we view the cylinder groups as horizontal units going from left to right.

The HFS file system is made up of multiple cylinder groups, where the number of cylinder groups is dependent on the size of the file system. In the slide, we assume the HFS file system takes the whole disk, therefore, there are N cylinder groups in the sample file system. Typically, they are numbered from 0 to $N-1$.

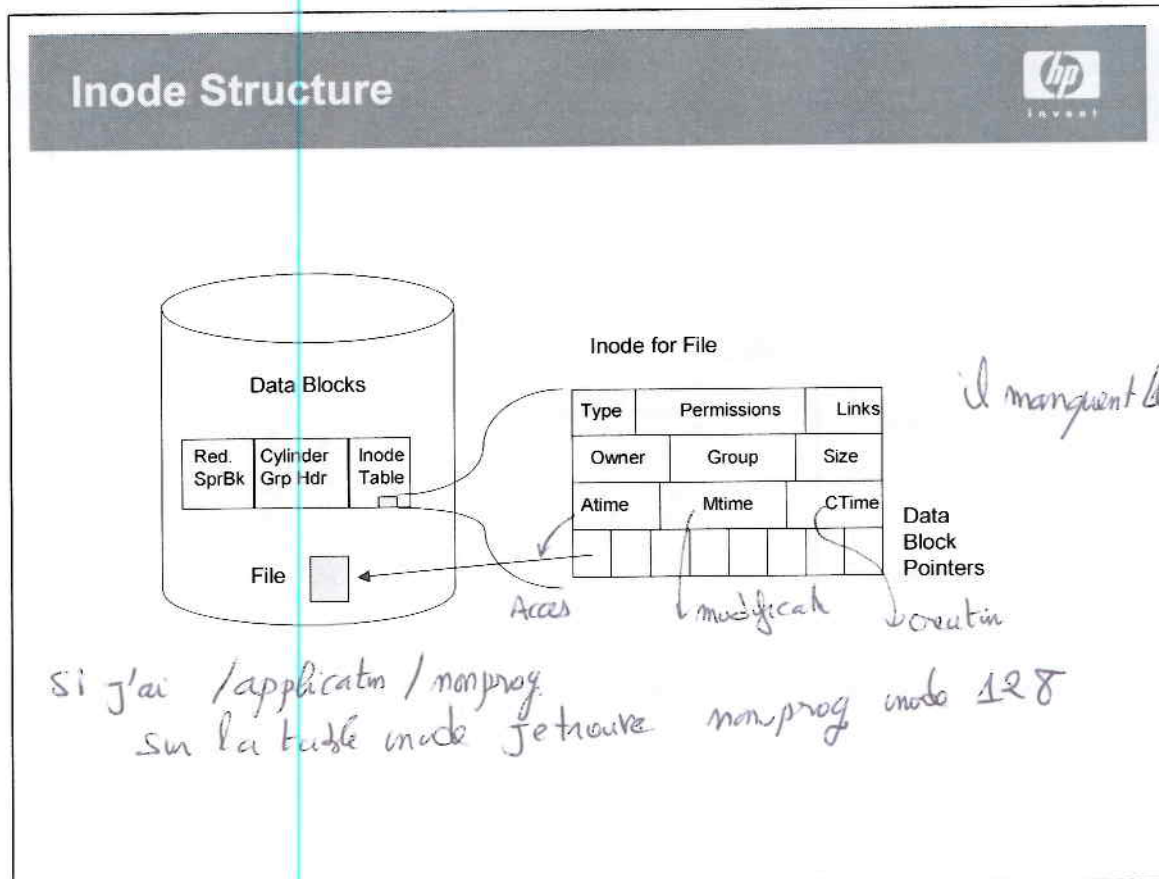
A critical data structure contained with every HFS file system is the **primary superblock**. The primary superblock is located at the start of every HFS file system at the start of the first cylinder group, and contains the critical header information for the HFS file system. Data structures contained within the superblock include the free block list, the mount flag, the starting address of each cylinder group, and much more.

Internal Cylinder Group View

Within each cylinder group, the following data structures exist:

Data blocks	The data blocks are where files are stored within the cylinder group. The data blocks are distributed in such a way that a portion of the data blocks come before the cylinder group header structures and the rest come after the cylinder group header structures. This ensures that the cylinder group header structures are randomly placed throughout the cylinder groups.
Redundant Superblock	A redundant copy of the primary superblock is contained within each cylinder group. These redundant copies are kept to protect against the loss of the primary superblock. The locations of the redundant superblocks can be viewed by displaying the contents of the <code>/etc/sbtab</code> file. Should the primary superblock become lost or corrupted, the file system could still be recovered by executing the <code>fsck</code> command and specifying the location of one of the alternate superblocks.
Cylinder Group Header	The cylinder group header contains the header information for the cylinder group. This information includes the free blocks within the cylinder group, the starting addresses of the inode tables for that group, and a list of free inodes for the local inode table.
Inode Table	The inode table contains all the inodes (file header structures) for files located within the cylinder group. Every file within a file system is managed by an inode, which describes the attributes and location of the file. The inode table is divided into equal-sized sections and a section is stored in each cylinder group. Inodes within a cylinder group point to files usually contained within the same cylinder group.

9-2. SLIDE: Inode Structure



Student Notes

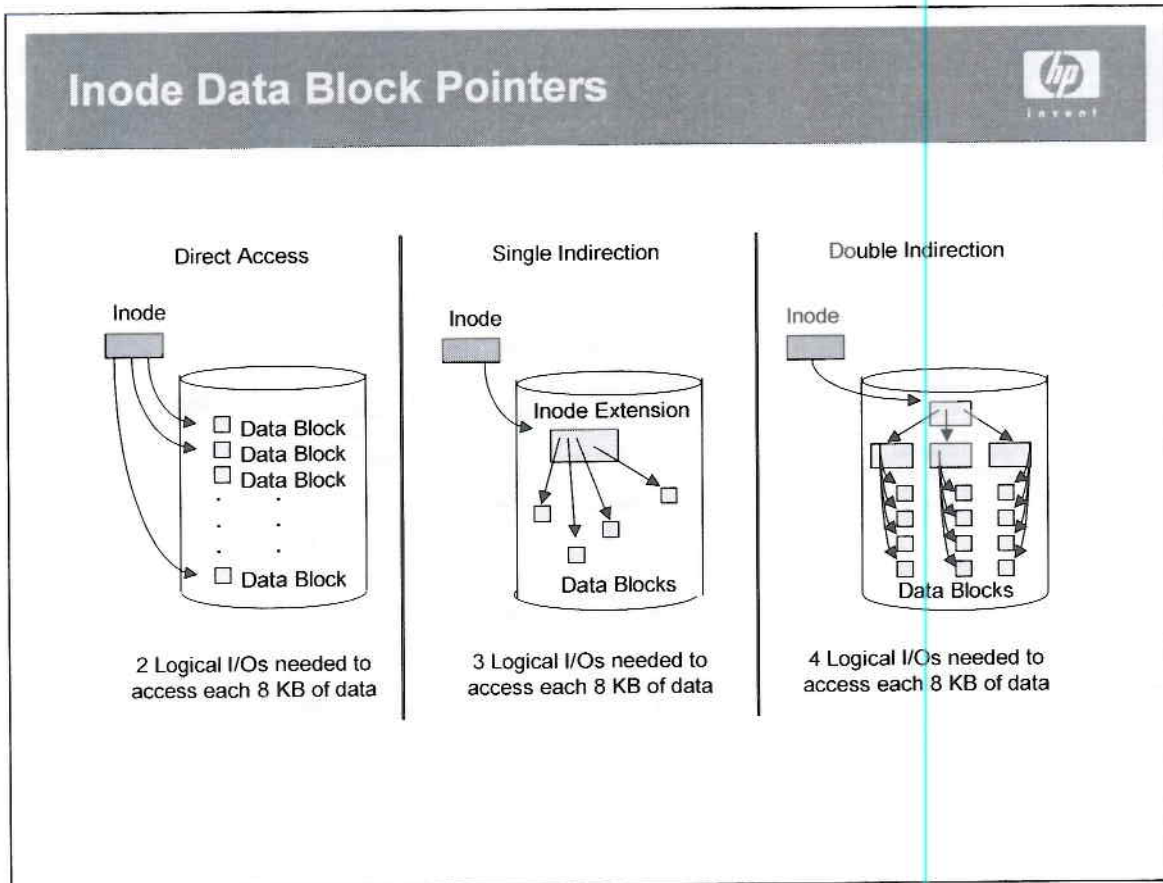
An inode contains all the header information for a particular file. Every file has a corresponding inode, usually located within the same cylinder group as the file.

Fields contained within the inode include:

- File type
- File access permissions
- Number of hard links to the file
- Owner and group of the file
- Size of the file in bytes
- Time stamps (file access, file modification, inode changes)
- Data block pointers (direct and indirect)

NOTE: Although the size of the inode differs from one type of file system to another, the basic types of data contained is virtually the same, the main differences are in the data pointer structures.

9-3. SLIDE: Inode Data Block Pointers



Student Notes

One of the structures within each HFS inode is the array of data block pointers that reference the data blocks within the file. The size of the data block pointer array is 15 entries, meaning there are a maximum of 15 file system block addresses within the array.

The first 12 addresses within the data block pointer array are "direct access" addresses. The thirteenth entry is a "single indirection" block address, the fourteenth is a "double indirection" block address, and the fifteenth (and last) entry is a "triple indirection" block address.

Direct Access

A direct access address points directly to a file's data block. When accessing a file using a "direct access" address, a minimum of two logical I/Os are needed: one I/O to access the file's inode (containing the direct access address), and one I/O to access the file's corresponding data block.

Single Indirection

Single indirection implies the address within the inode references a block on disk that acts as an inode extension block. The inode extension block, in turn, contains addresses that point to the file's corresponding data blocks.

It should be noted that three logical I/Os are needed to access a file's data blocks using single indirection: one I/O for the file's inode, one I/O for the inode extension block, and one I/O for the data block itself.

Double Indirection

Double indirection means access to a file's data blocks require going through two inode extension blocks. The first inode extension block references the address of a second inode extension block, which contains addresses referencing the file's **data blocks**. Double indirection is needed only for files above 16 MB (with a default block size of 8KB).

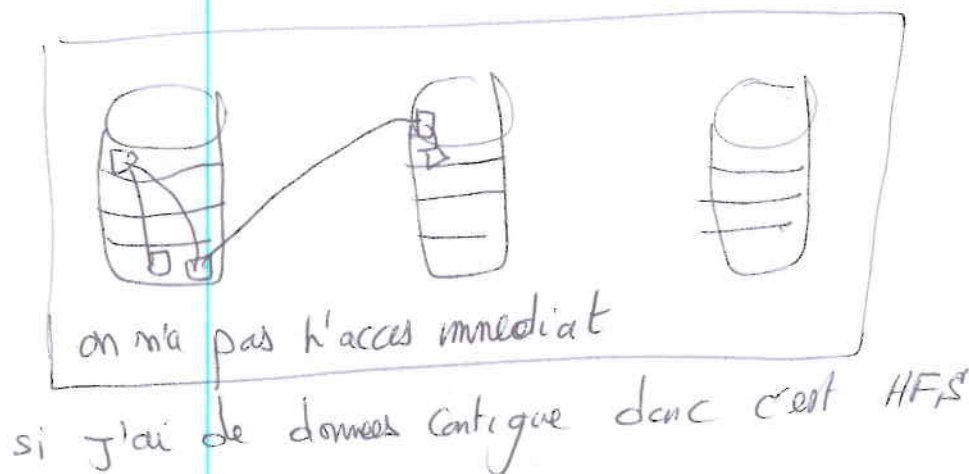
When accessing files requiring double indirection, a total of four logical I/Os are required: an I/O for the file's inode, an I/O for each of the two inode extension blocks, and an I/O for the file's data block.

Triple Indirection

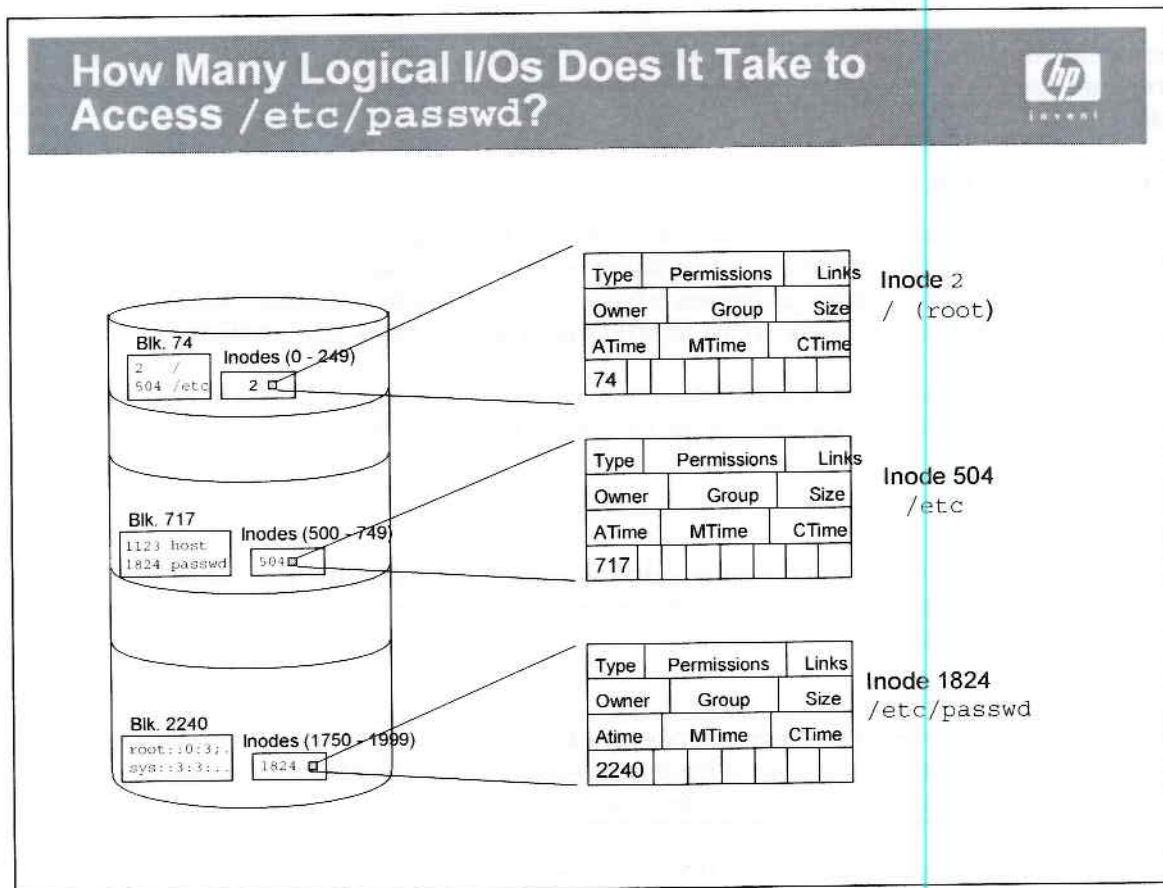
Triple indirection (not shown on the slide) adds one more level of indirection when accessing a file's data blocks. Triple indirection is only needed to access files larger than 32 GB (with a default block size of 8KB).

NOTE: Every level of indirection adds an additional logical I/O when accessing the file's data. In the case of triple indirection, five logical I/Os are needed compared to two I/Os for direct access data blocks.

As you can see, the performance of an HFS file system tends to favor small files (12 blocks or less), and tends to penalize large files that have to use single, double, or even triple indirection. You can delay this performance degradation somewhat, by building the file systems with larger block sizes. (More on that later in the module.)



9-4. SLIDE: How Many Logical I/Os Does It Take to Access /etc/passwd?



Student Notes

The above slide illustrates how a file within the HFS file system is accessed. It may surprise some people when they find out how many logical I/Os are needed to access the /etc/passwd file.

Starting from the Top

When the full pathname of a file is specified for access (as in /etc/passwd), the kernel starts with the only inode it knows: the inode of the root directory of the root file system. Inode number 2 is always the inode of the root directory of any file system. "/" symbolizes (in the kernel) the root directory of the root file system.

Using the slide as an example, after reading inode 2 of the root file system (first logical I/O), the kernel discovers that the contents of the root directory (the listing of the files contained in that directory) are located at file system block 74. Upon reading block 74 (second logical I/O), the names of the files in the root directory and their corresponding inode numbers are known. Directories are primarily listings of file names and the numbers of the inodes that manage them.

From this information, the kernel discovers the inode for the `etc` directory (in `"`) is 504. Inode 504 is then read (third logical I/O) and from that the kernel learns the `etc` directory is located at file system block 717. Block 717 is read (4th logical I/O) and the file names and inodes contained within that directory are now known.

One of the entries within block 717 is the `passwd` file and its corresponding inode number 1824. The inode 1824 is read (5th I/O), and from this the kernel finally learns that block 2240 is the one that contains the contents of the `/etc/passwd` file. Block 2240 is read (6th I/O) and the kernel finally has the data it set out to access.

So, the answer to the question at the top of the slide, "How many logical I/Os does it take to access `/etc/passwd`?" is ... 6.

DLNC qui appartient au buffer cache : elle contient ces inodes déjà ouverts et dès que je ferme le fichier le DLNC vide ces inodes

9-5. SLIDE: File System Blocks and Fragments

File System Blocks and Fragments



Fragment {	FileA	FileD	FileE	FileF	End of Disk
	FileB	FileD	FileE	FileF	
	FileB	FileD	FileE	FileF	
	FileC	FileD	FileE	FileF	
	FileC		FileE	FileF	

File System Block

FileA, FileC, and FileD grow by 1 fragment

	FileD	FileE	FileF	End of Disk
FileB	FileD	FileE	FileF	
FileB	FileD	FileE	FileF	
	FileD	FileE	FileF	
	FileD	FileE	FileF	
FileA		FileC	FileF	
FileA		FileC		
		FileC		

Student Notes

The concept of blocks and fragments was introduced when the HFS file system was designed. There is always a tradeoff when managing a resource based on a fixed allocation unit size (the file system "block" in this case). If the block size is large we can manage them with fewer pointers (system overhead) but if it is too large there is an opportunity for inefficient utilization of the resource (very small files still require a block). In the case of the HFS file system this concern was addressed by making the block capable of uniform subdivision. The fragment was created for this purpose.

Definitions

- Sector** A sector is the smallest unit of space addressable on the physical disk. The sector size is used when the disk is formatted to appropriately place timing markers on the platter. The default sector size for HP-UX and most UNIX systems is 512 bytes.
- Fragment** A fragment is the increment in which space is allocated to files within the HFS file system. The default fragment size is 1 KB. This can be tuned when the HFS file system is initially created. Allowable sizes are 1K, 2K, 4K, and 8K.

File System Block A file system block is the minimum amount of data transferred to/from the disk when performing a disk I/O on an HFS file system. The default file system block size is 8 KB. This can be tuned when the HFS file system is initially created. Allowable sizes are 4K, 8K, 16K, 32K, and 64K.

Example — Top Half

The top half of the slide shows the allocation of disk space when the following six files are created (assuming only 5 file system blocks are free within the HFS file system).

File A (size 1 KB): The kernel searches for the first free fragment. On the slide, the first fragment in the first file system block is allocated.

File B (size 2 KB) The kernel searches for the first 2-KB continuous fragment that is available. This is in the same file system block in which **FileA** was allocated. The fact that **FileA** has already been allocated in this file-system block does not matter. *Multiple files can be allocated within the same file system block.* The first basic rule is: Best fit on “close”.

File C (size 2 KB): The kernel searches for the first 2-KB continuous fragment available. This is in the same file system block as **FileA** and **FileB**. Hence, **FileC** is allocated 2 KB from this same file-system block. *If any of these three files are accessed, then all three files are read into the file system buffer cache as a single unit.*

File D (size 4 KB): The kernel searches for the first four contiguous 1KB fragments available (within the same file system block). This is in the second file system block. The kernel does not allocate 3 fragments from the first file system block and 1 fragment from the second file system block, because that would require two logical I/Os to read in the entire 4 KB. This is inefficient, as only one I/O is required if the file is contained within the same file system block. The second basic rule is: *If the size of a file is 8 KB or less, the kernel will fit the entire file within a single file-system block.*

File E (size 5 KB) and **File F** (size 6 KB):

The kernel searches for the first available file-system block that can hold the entire file. On the slide, **FileE** is allocated in file system block 3, and **FileF** is allocated in file system block 4.

Example — Bottom Half

The bottom half of the slide illustrates how the growth of three files affects allocation within the HFS file system.

FileA (1KB -> 2KB): When **FileA** grows, it cannot grow into the next fragment because **FileB** is occupying this spot. Therefore, the kernel relocates **FileA** to the first free 2 KB that is within the *same* file system block. (Why transfer another block into memory, at this point?)

FileC (2KB -> 3KB): When **FileC** grows, it *cannot* grow into the next fragment, because **FileA** is now in that spot. Therefore, the kernel relocates **FileC** to the first free 3 KB that is in a *different* file system block. (It can no longer fit into the first block). It selects block three because that block has a space exactly suited for the three-block **FileC**. The third basic rule is: if a file owns multiple fragments within the same block, they must be contiguous.

FileD (4KB -> 5KB): When **FileD** grows, it simply grows into the next fragment because it is still free.

9-6. SLIDE: Creating a New File on a Full File System

Creating a New File on a Full File System


New FileG (4 KB) is created

	FileD	FileE	FileF	FileG	
FileB	FileD	FileE	FileF	FileG	End
FileB	FileD	FileE	FileF	FileG	
	FileD	FileE	FileF	FileG	of
	FileD	FileE	FileF		
FileA	FileC		FileF		Disk
FileA	FileC				
	FileC				

What happens when new FileH (1 KB) is created?

Student Notes

As an HFS file system becomes full, the performance impact of creating a new file becomes significant. This is due to the behavior of the kernel when creating a new file: *When a new file is created on HFS file systems, the kernel tries to allocate a block-sized buffer in buffer cache for the file to grow into. Upon the file being closed, the kernel allocates the file's fragments to an already allocated file system block, if possible.*

FileG Is Created

In the example on the slide, **FileG** is opened/created as a new file. Not knowing the size to which **FileG** will grow, the kernel allocates a block-sized buffer in buffer cache for **FileG** to grow into. When **FileG** is closed, the kernel searches for a set of four contiguous 1KB fragments in a block. Since there are no shared blocks that have four contiguous fragments, the file is written to a new, empty block.

What Happens When FileH Is Created?

The impact of creating new files on a full file system can be seen when **FileH** is created. When **FileH** is opened for creation, the kernel allocates a block-sized buffer in buffer cache.

Module 9
HFS File System Performance

As it turns out, **FileH** is closed after writing only 1 KB worth of data. Upon closure, **FileH** is moved to file system block 1, first fragment.

NOTE:	Performance on HFS file systems typically degrades when free space falls below 10%, due to the length of time it takes to find free file system blocks for new files. For this reason, it is recommended that MINFREE always be 10% or greater, even for large file systems (greater than 4 GB).
-------	--

The fourth basic rule is: No fragment belonging to another file will be moved to make room for this file.

9-7. SLIDE: HFS Metrics to Monitor — Systemwide

HFS Metrics to Monitor — Systemwide



- Utilization of the file systems
- File system I/O queue lengths
- Amount of physical I/O to the file systems
- File system free space
- Open files for each process

Student Notes

When monitoring disk I/O activity, the main metrics to monitor are:

- **Percent utilization of the file systems:** As utilization of the file system 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 file system is 50% busy, than it does when the file system is idle.
- **Requests in the file system I/O queue:** The number of requests in the file system I/O queue is one of the best indicators of a file system performance problem. If the average number of requests is three or greater, then requests are having to wait in the queue longer than the amount of time needed to service those requests.
- **Amount of physical I/O:** If the amount of file system activity is high, it is important to investigate on which file system the activity is occurring.
- **File system free space:** As an HFS file system becomes full (greater than 90%), it takes longer and longer to find an available free fragment for a new file or to grow an existing file. This creates additional disk activity, leading to slow file system performance.

Module 9
HFS File System Performance

- **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, (hit <CR> frequently, then inspect how quickly the offset to each file changes) consider relocating these files to other disks that are less busy.

9-8. SLIDE: Activities that Create a Large Amount of File System I/O

Activities that Create a Large Amount of File System I/O



- File writes on an almost full file system
- Long, inefficient PATH variables
- Deep subdirectory structures
- Accessing large files sequentially with a small READ block size
- Accessing many files on a single disk

Student Notes

Common causes of disk-related performance problems are shown on the slide.

- Full file system cause excessive I/O due to locating free fragments.
- Long, inefficient PATH variables cause excessive directory I/O (especially when the command is found in the last directory within the PATH variable).
- Deep subdirectories cause lots of logical I/Os (two logical I/Os for each subdirectory in the full path name).
- Sequential file access, with a small file system block size, causes excessive amounts of physical I/O.
- Accessing lots of files on one file system, versus many, creates an imbalance of utilization. This leads to performance problems with the busy file systems and under utilization with the others.

9-9. SLIDE: HFS I/O Monitoring bdf Output

HFS I/O Monitoring bdf Output					
					
# bdf					
Filesystem	kbytes	used	avail	%used	Mounted on
/dev/root	81920	38018	40901	48%	/
/dev/vg00/lvol1	47829	22403	20643	52%	/stand
/dev/vg00/lvol6	286720	257116	28003	90%	/usr
/dev/vg00/lvol4	360448	346127	13444	96%	/opt
/dev/dsk/c0t4d0	1177626	1113204	0	100%	/disk
/dev/vg00/lvol7	122880	102098	19257	84%	/var
/dev/vg00/lvol5	53248	22589	28549	44%	/tmp

Student Notes

The **bdf** report shows how much file system space is being used (and how much is free) for all file systems currently mounted on the system. The key fields are:

avail Indicates the amount of disk space available on the file system (in KB).

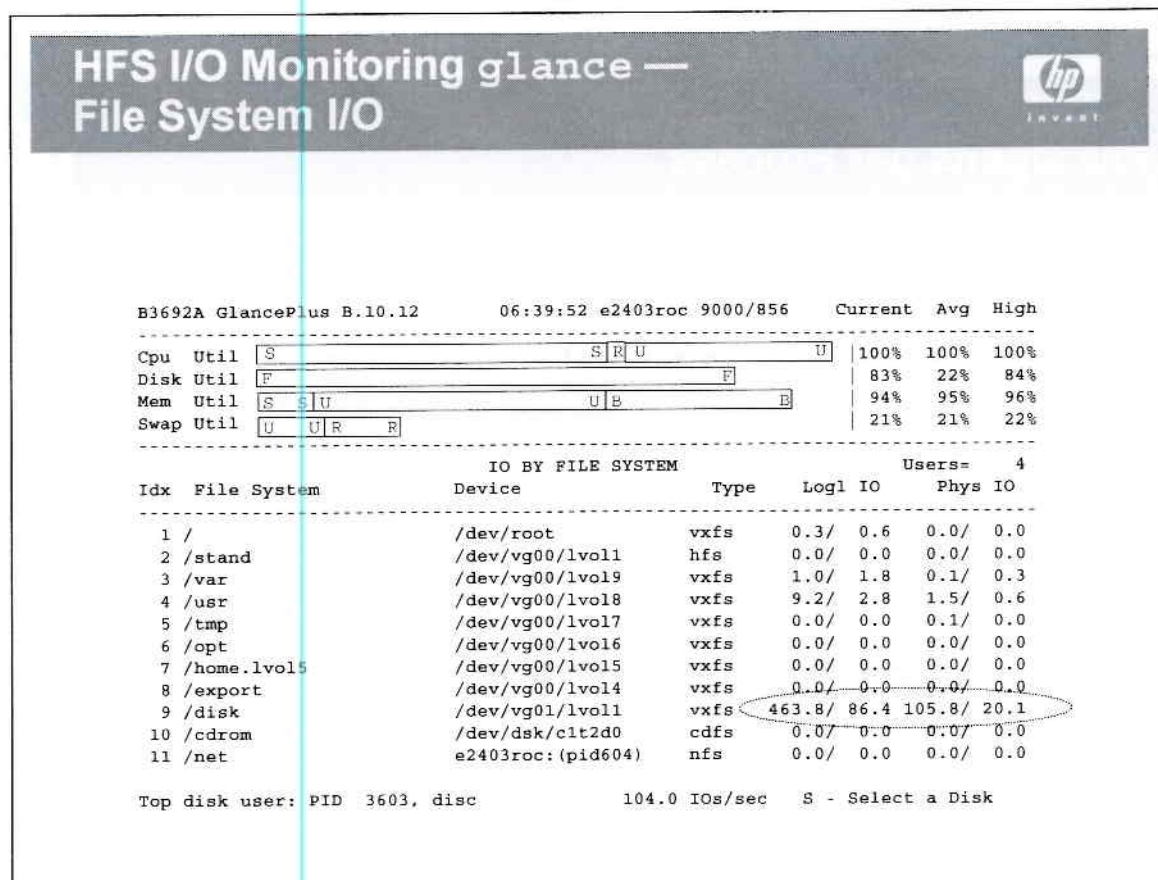
%used Indicates the percentage of disk space used.

The slide shows there are three file systems with 90% usage or more, and one of the file systems is at 100% utilization. Recall that when an HFS file system becomes full, performance on that file system suffers due to fragments being moved.

The good news is that the amount of free space which is being held back by the file system parameter, MINFREE, is already subtracted from the values. In fact, if you compare the **kbytes**, **used**, and **avail** columns, you'll see that something is missing. **used + avail** do not add up to be **kbytes**. The difference is MINFREE.

For example, look at **/stand**. Clearly, $22403 + 20643$ does not equal 47829. In fact, $22403 + 20643$ divided by 47829 equals 90%, indicating that MINFREE must be set to 10% for this file system.

9-10. SLIDE: HFS I/O Monitoring glance — File System I/O



Student Notes

The **g**lance file system I/O report (**i** key) shows activity on a per file system basis. Only total I/O activity (not reads versus writes) is shown with this report.

This report is similar to the logical volume report (discussed in the previous module) except this report shows logical I/O compared to physical I/O, and does not distinguish between read and write activities. The logical volume report shows reads compared against writes, but does not distinguish between logical and physical activities.

From the report on the slide, we note that all the file system activity is being performed against one file system.

Note: The file system I/O report shows I/O activity for all types of mounted file systems, including CDFS file systems and NFS-mounted file systems.

To determine whether I/O activity is occurring against a file, enter the open file report for a particular process, and press <CR> multiple times in succession. Watch the `offset` field for each file. If the `offset` field is constantly changing, it indicates the file is currently being accessed.

Performance Scenario

A system is experiencing slow performance due to high file system utilization. Upon further investigation, not all file systems are heavily utilized. In fact, some show no activity at all.

By sorting the processes within glance by disk I/O activity, then selecting those processes to obtain further details, you can determine which files are getting the majority of the activity. To take advantage of the underutilized file system, move the heavily accessed files to this file system and create a symbolic link to the file from its original location, thereby removing a heavily accessed file from a busy file system and putting it on an underutilized file system.

9-12. SLIDE: Tuning a HFS I/O-Bound System — Tune Configuration for Workload

Tuning an HFS I/O-Bound System — Tune Configuration for Workload



Tune the following parameters, based on workload:

- File System Block and Fragment Sizes
- Blocks per cylinder group (`maxbpg`)
- File system mount options
- The `mkfs` options when creating the file system
- The `tunefs` options can modify parameters on existing file systems

Tune other configurations, based on workload:

- Optimize `$PATH` variables
- Use flat directory structures when possible
- Ensure sufficient freespace exists on file systems

Student Notes

Every workload and every application is different. Each has different resource requirements and each places different demands on the system. There is no one configuration that is optimal for all applications. For example, CAD-CAM application stress memory (and graphics); accounting applications that do forecasting stress CPU; NFS-based applications stress the disks (and the network); and RDBMS applications stress all resources.

File System Blocks and Fragments

Tips and notes for choosing the sizes of file system block and fragments follow:

Fragments

- Fragment sizes can be 1, 2, 4, or 8 KB in size.
- Fragments can be 1/8, 1/4, 1/2, or equal to the file system block size.
- For large files which are opened and closed a lot during their growth, large fragments are recommended.
- For file systems with lots of small files, small fragments are recommended.

File System Blocks

- File system blocks sizes can be 4, 8, 16, 32, or 64 KB.
- For file systems with large files, large file system blocks are recommended.
- For file systems with large files, increase **maxbpg** (maximum blocks per group).
- For applications which perform a lot of sequential I/O (with read-aheads and write-behinds), large file system blocks are recommended.

HFS Mount Options

The mount options affect performance by specifying when files on the file system are updated. These options can be specified in the options column of the `/etc/fstab` file.

The HFS-specific mount options include:

behind	Enable, when possible, asynchronous writes to disk. This is the default for workstations. It does not use the sync daemon.
delayed	Enable delayed or buffered writes to disk. This is the default for servers. It does use the sync daemon.
fs-async	Enable relaxed (asynchronous) posting of file system metadata (changes to the superblocks, inodes, etc.). This option may improve file system performance, but increases exposure to file system corruption in the event of power failure.
no_fs_async	Force rigorous (synchronous) posting of file system metadata to disk. This is the default.

mkfs Options

mkfs is usually not executed directly, but is called by **newfs -F hfs** instead. File system tuning is best accomplished when the file system is created. The workload for a file system should be well understood and dedicated before serious attempts are made to tune one. Many options are also dependent on the type of physical device on which a file system is being created.

The HFS specific options include:

size	The size of the file system in DEV_BSIZE blocks (the default is the entire device).
largefiles	The maximum size of a file can be up to 128 GB.
nolargefiles	The maximum size of a file will be limited to 2 GB.
ncpg	The number of cylinders per cylinder group (range 1-32, the default is 16).
minfree	the minimum percentage free disk space reserved for non-root processes (default is 10%). Beginning with HP-UX 10.20 the bdf command does not conceal this free space and as a result will report free disk space accurately. This means that a file system cannot show 111% utilization anymore.

nbpi	The number of bytes per inode. This value determines how many inodes are allocated given a file system of a certain size. (The default is 6144.)
-------------	--

tunefs Options

Some parameters can be changed once the file system has been created, with `tunefs (1m)`. There are `minfree` and `maxbpg`. `minfree` is explained above.

maxbpg	The maximum number of data blocks that a large file can use out of a cylinder group, before it is forced to continue to grow in a different cylinder group. This value does not apply to any file which size is 12 blocks or less.
---------------	--

tunefs can also be used to display the contents of an HFS file system:

```
# tuneufs -v /dev/.../...
```

Other Configurations

Optimize \$PATH

The PATH variable in a user's environment specifies a list of directories to search when a command is entered. Having an excessive number of directories or duplicate directories to search can increase disk access, particularly when the user makes a mistake typing a command. This problem can be greatly exacerbated if the user's PATH variable contains directories that are mounted automatically with the NFS automount utility, causing the network mount of a file system because of a typographical error.

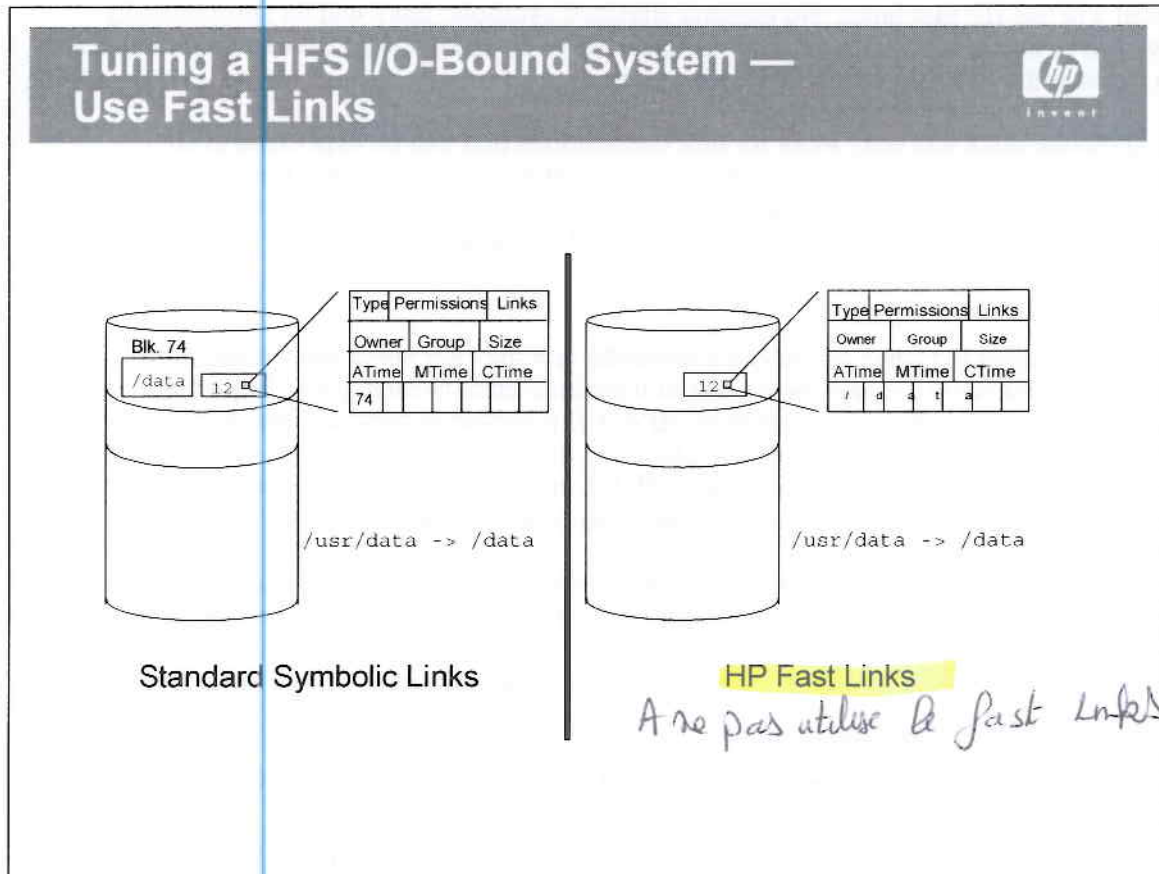
Use Flat Directory Structures

Long directory path names create more work for the system because each directory file and its associated inode entry require a disk I/O in order to bring them into memory. Recall that six logical I/Os were required to read the `/etc/passwd` file. Conversely, you don't want thousands of files in the same directory, as it would take many I/O operations to read and search the directory.

Ensure Sufficient Freespace

As the file system becomes full (greater than 90%), the kernel begins to take longer and longer to find available free fragments. The algorithm gets very lengthy when the file system free space falls below 10%. Of course, if you do not have any files that grow and you are not adding any new files, this would waste 10% of your file system free space for no reason.

9-13. SLIDE: Tuning a HFS I/O-Bound System — Use Fast Links



Student Notes

There are two ways symbolic links can be stored on HFS file systems.

Standard Symbolic Links

Standard symbolic links are implemented in the same way as they are on other UNIX systems. The inode for the symbolic link points to a data block on disk, and the contents of the data block contain the name of the file being referenced by the symbolic link.

In the example on the slide, `/usr/data` is the symbolic link with an inode number of 12. The contents of inode 12 contain an address pointer to data block 74, and the contents of data block 74 contain the name of the file being referenced (in the example, `/data`). Two logical I/Os are required to resolve the symbolic link, one I/O to retrieve the inode and one I/O to retrieve the data block containing the referenced name.

HP Fast Links

HP fast links allow symbolic links to be resolved with one logical I/O instead of two. HP fast links store the name of the referenced file in the inode of the symbolic link itself, rather than in a data block that the inode references. In the example, when the inode (12) of the symbolic link is retrieved, the contents of the inode contain the name of the referenced file.

HP fast links can be configured by setting the tunable OS parameter `create_fastlinks` to 1, and recompiling the kernel. Upon booting from the new kernel, all future symbolic links created will use HP fast links. No existing standard symbolic links will be automatically converted to fast symbolic links. The standard symbolic links would have to be removed and then recreated to convert them.

Fast symbolic links will only work for link destinations that can be expressed in 59 characters or less as this is the limit of the space within the inode where the fast link information is stored. If a symbolic link contains more than 59 characters, it will be stored as a standard symbolic link, regardless of the value of `create_fastlinks`.

Transition Links

Saving one logical I/O when accessing a symbolic link may not seem significant, until considering that HP-UX makes heavy use of transition links (which are an implementation of symbolic links). Transition links allow an HP-UX file system to contain older 9.x directory paths. The 9.x directory names are symbolic links that point to the correct, current location (for example, `/bin > /usr/bin`). Many HP-UX installations have applications (including HP-UX applications), which rely on and make heavy use of transition links.

A quick performance gain for all HP-UX systems is to convert these transition links from standard symbolic links to HP fast links. The procedure for making this conversion is:

1. Recompile the kernel to use HP fast links (i.e. set the `create_fastlinks` to 1).
2. Shut down and reboot the system.
3. Execute `tlremove` to remove all the transition links from the system. Over 500 links will be removed.
4. Execute `tlinstall` to reinstall (that is, recreate) the transition links. When the links are reinstalled, they will be created with HP fast links.

9-14. LAB: HFS Performance Issues

Directions

The following lab illustrates a number of performance issues related to HFS file systems.

1. A 512 MB HFS file system is required for this lab. Use the `mount` and `bdf` commands to determine if such a file system is available.

```
# mount -v
# bdf
```

If there is no such HFS file system available, create one using the commands below:

```
# lvcreate -n hfs vg00
# lvextend -L 512 /dev/vg00/hfs /dev/dsk/cXtYdZ (second disk)
# newfs -F hfs /dev/vg00/rhfs
# mkdir /hfs
# mount /dev/vg00/hfs /hfs
```

2. Copy the lab files to the newly created HFS file system.

```
# cp /home/h4262/disk/lab1/disk_long /hfs
# cp /home/h4262/disk/lab1/make_files /hfs
```

Next, execute the `make_files` program to create five 4-MB ASCII files.

```
# cd /hfs
# ./make_files
```

3. Purge the buffer cache of this data, by unmounting and remounting the file system.

```
# cd /
# umount /hfs
# mount /dev/vg00/hfs /hfs
# cd /hfs
```

Module 9

HFS File System Performance

4. Time how long it takes to read the files with the `cat` command. Record the results below:

```
# timex cat file* > /dev/null
```

real:

user:

sys:

5. In a 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?

6. Performance tuning — recreate the file system with larger fragment and file system block sizes.

Tuning the size of the fragments and file system blocks can improve performance for sequentially accessed files. The procedure for creating a new file system with customized fragments of 8 KB and file system blocks of 64 KB is shown below:

```
# lvcreate -n custom-lv vg00
```

```
# lvextend -L 512 /dev/vg00/custom-lv /dev/dsk/cXtYdZ
```

```
# newfs -F hfs -f 8192 -b 65536 /dev/vg00/rcustom-lv
```

```
# mkdir /cust-hfs
```

```
# mount /dev/vg00/custom_lv /cust-hfs
```

7. Copy the lab files to the customized HFS file system, execute the `make_files` program, and purge the buffer cache.

```
# cp /hfs/disk_long /cust-hfs
```

```
# cp /hfs/make_files /cust-hfs
```

```
# cd /cust-hfs
```

```
# ./make_files
```

```
# cd /
```

```
# umount /cust-hfs
```

```
# mount /dev/vg00/custom-lv /cust-hfs
```

```
# cd /cust-hfs
```

8. Time how long it takes to read the files with the **cat** command. Record the results below:

```
# timex cat file* > /dev/null
```

real:

user:

sys:

How do the results of step 8 compare to the default HFS block and fragment results from step 4?

9. Performance tuning — change file system mount options.

The manner in which the file system is mounted can impact performance. The **fsasync** mount option can improve performance, but data (metadata) integrity is not as reliable in the event of a crash, and **fsck** could run into difficulties.

```
# cd /
```

```
# umount /hfs
```

```
# mount -o fsasync /dev/vg00/hfs /hfs
```

```
# cd /hfs
```

10. In a 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 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?

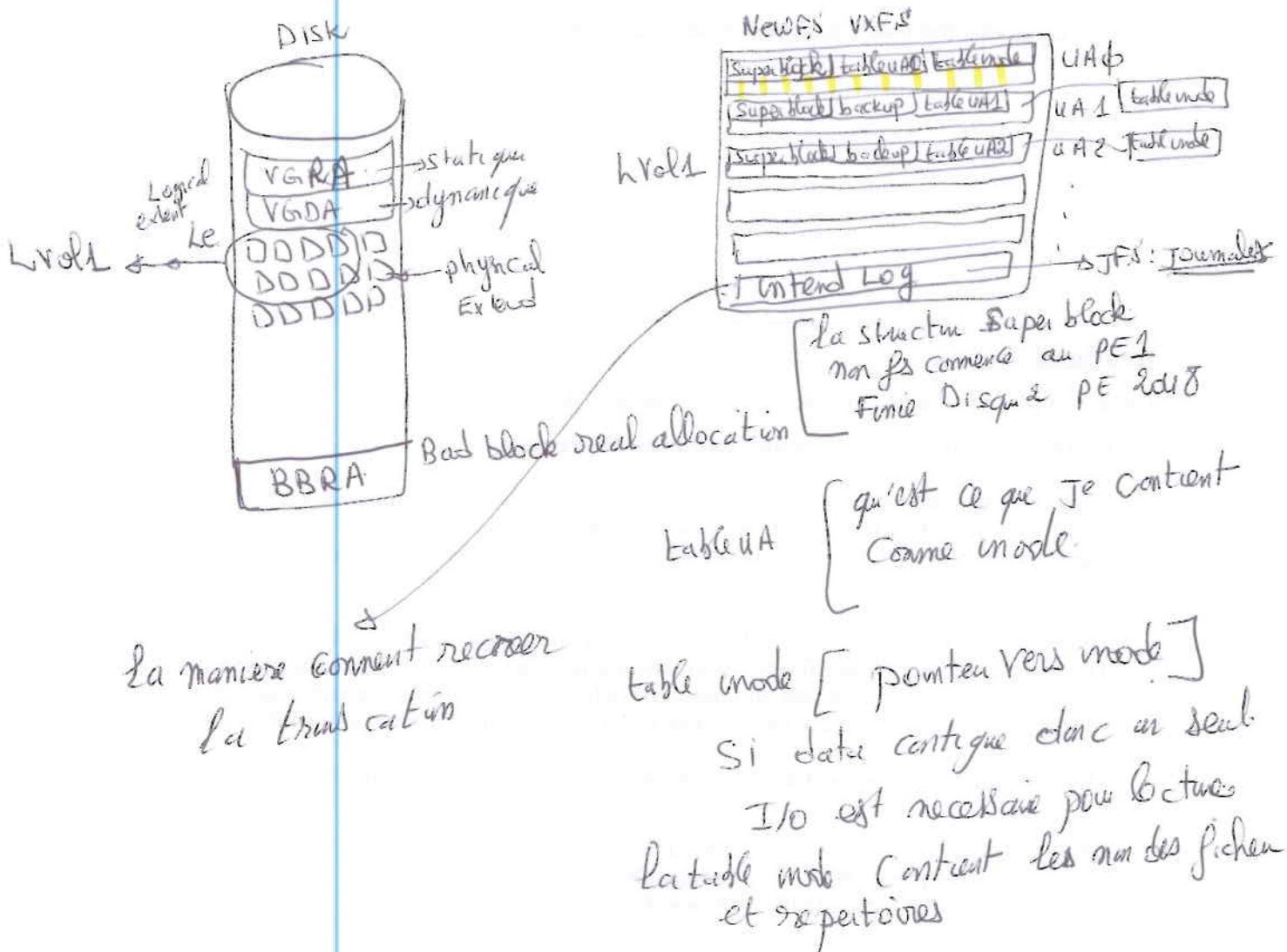
How do the results of step 10 compare to the default mount options in step 5?

Module 9
HFS File System Performance

Module 10 — VxFS Performance Issues

- 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

pour connaître la taille du
block
mkfs -m /dev/vg01/rvold
et regarder
bsize = 8192
~~ou avec Fscadm -b~~
ou avec Fstype -v
F-Fsize



10-1. SLIDE: Objectives

Objectives



Upon completion of this lesson, you will be able to:

- 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 I/O
- Identify JFS tuning parameters
- Understand and control fragmentation issues
- Evaluate the overhead of online backup snapshots

Student Notes

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

- **Understand JFS Structure and version differences**

These course notes are based on the JFS Version 3.5 file system, built on Version 4 disk layout. The next few slides will describe the basic differences between versions and relate them to HP-UX releases. HP JFS 3.5 and HP OnlineJFS 3.5 are available for HP-UX 11i and later systems.

The standard (base) version of HP JFS has been bundled with HP-UX since release 10.01. The "advanced" HP OnlineJFS is a purchasable product with additional administrative features for higher availability and tunable performance. These notes will make clear which features belong to the base product and which belong to the OnlineJFS version.

The Operating Environment delivery model of HP-UX 11i includes JFS as follows:

HP-UX 11i OE	BaseJFS 3.3
HP-UX 11i Enterprise OE	OnlineJFS 3.3
HP-UX 11i Mission Critical OE	OnlineJFS 3.3

You can download JFS 3.5 for HP-UX 11i for free from the HP Software Depot (<http://www.software.hp.com>), or you can request a free JFS 3.5 CD from the Software Depot. You can purchase HP OnlineJFS 3.3 (product number B3929CA for servers and product number B5118CA for workstations) for HP-UX 11.0 or HP-UX 11i from your HP sales representative. JFS 3.5 is included with HP-UX 11i systems.

- **Explain how to enhance JFS performance**

The HFS file system uses block based allocation schemes, which provide adequate random access and latency for small files but limit throughput for larger files. As a result, the HFS file system is less than optimal for commercial environments.

VxFS addresses this file system performance issue through an alternative allocation scheme and increased user control over allocation, I/O, and caching policies.

- **Set Block Sizes to improve performance**

It is often advantageous to match the block size of a file system to the I/O size of the application. We will show you how!

- **Set Intent Log size to improve performance**

The JFS intent log provides for rapid `fsck` recovery after a system crash. In general the intent log is not protecting your data; the focus is on structural integrity and not data integrity! Fast `fsck` comes at a price and that price is performance. Setting the correct intent log size is important as it cannot be changed once a file system is created.

- **Understand and manipulate synchronous and asynchronous I/O**

Programmers and data base providers do different types of I/Os to obtain the best possible balance between data integrity and performance. We will investigate all the “gray” areas and tune the JFS file system to meet our administrative and performance goals which might be quite different to those of the programmer!

- **Identify JFS tuning parameters**

The JFS is tunable through mount options, the command line, configuration files and kernel parameters. We will learn where and how to tune.

- **Understand and control fragmentation issues**

The extent based file allocation design of JFS is ideal for performance of large files. One weakness of this approach is the potential fragmentation of files and free space over the life of the file system. In general this will only occur in dynamic work file orientated JFS file systems (e.g. a mail server) and is unlikely in fixed “large file” file systems where major I/O rates occur to static files (e.g. a data base). We will investigate ways of measuring and fixing fragmentation.

- **Evaluate the overhead of online backup snapshots**

OnlineJFS supports online backups via snapshot mounts. We will discuss the performance issues involved when working with snapshots.

10-2. SLIDE: JFS History and Version Review

JFS History and Version Review



- JFS introduced in 1995 with HP-UX 10.01
- Version 2 structure at introduction
- Version 3 structure at 10.20 allows 1TB files
- Version 4 structure at 11.00 allows more tunable controls and supports ACLs
- Do not use V4 structure on 11.00 for /, /usr, /opt, /var
- vxupgrade(1M) tool can migrate up through versions (not down!)
- 11i delivers JFS 3.5 software on V4 structure
- Differences between Base JFS 3.5 and OnlineJFS 3.5

Student Notes

The HP-UX Journaled File System (JFS) was introduced by HP in August, 1995, on the HP-UX 10.01 release. The journaled file system attempts to improve on the high-performance file system (HFS) by offering the following enhancements:

- Extent-based allocation of disk space
- Fast file system recovery through an Intent Log
- Greater control and flexibility of file system behavior through new mount options and tunable options.

Disk Layout Versions

Version 1 The Version 1 disk layout was never used in HP-UX.

Version 2 The Version 2 disk layout has the following changes and features:

- Many internal JFS structures are dynamic files themselves.
- Internal “filesets” separate data files (User Fileset) from structural files (Structural Fileset).
- Allocation units now contain data and data map structures only, inode tables are elsewhere.
- inode allocation is dynamic and cannot run out.
- Optional support for quotas.

Version 3 The Version 3 disk layout offers additional support for:

- Files up to one terabyte
- File systems up to one terabyte
- Indirect inode extent maps can now address variant length file extents. V2 restricts all indirect extents to the size of the first indirect extent. Hence large files and sparse files possible with less overhead.

Version 4 Version 4 is the latest disk layout:

- The Version 4 disk layout supports Access Control Lists.
- The Version 4 disk layout does not include significant physical changes from the Version 3 disk layout. Instead, the policies implemented for Version 4 are different, allowing for performance improvements, file system shrinking, and other enhancements.
- HP-UX 11i with Version 4 layout now supports both files and file systems up to 2TB in size.

Table: Matching HP-UX version to JFS version

HP-UX Release	VxFS Version	Supported Disk Layouts	Default Disk Layout
10.10	2.3	2	2
10.20	3.0	2,3	3
11.00 with JFS 3.1	3.1	2,3	3
11.00 with JFS 3.3	3.3	2,3,4	3
11i v1	3.3	2,3,4	4
11i v2	3.5	2,3,4	4

vxupgrade (1M)

The **vxupgrade** command can upgrade an existing Version 3 VxFS file system to the Version 4 layout while the file system remains online.

vxupgrade can also upgrade a Version 2 file system to the Version 3 layout. See **vxupgrade(1M)** for details on upgrading VxFS file systems. You cannot downgrade a file system that has been upgraded.

NOTE: You cannot upgrade the root (/) or /usr file systems to Version 4 on an 11.00 system running JFS 3.3. Additionally, we do not advise upgrading the /var or /opt file systems to Version 4 on an 11.00 system. These core file systems are crucial for system recovery. The HP-UX 11.00 kernel and emergency recovery media were built with an older version of JFS that does not recognize the Version 4 disk layout. If these file systems were upgraded to Version 4, your system might have errors booting with the 11.00 kernel as delivered, or booting with the emergency recovery media.

Comparing Base and Advanced JFS

Table: Comparing Base and Online JFS

Feature	JFS 3.5	OnlineJFS 3.5
extent-based allocation	*	*
extent attributes	*	*
fast file system recovery	*	*
access control list (ACL) support	*	*
enhanced application interface	*	*
enhanced mount options	*	*
improved synchronous write performance	*	*
support for large files (up to two terabytes)	*	*
support for large file systems (up to two terabytes)	*	*
enhanced I/O performance	*	*
support for BSD-style quotas	*	*
unlimited number of inodes	*	*
file system tuning [<code>vxtnfs(1M)</code>]	*	*
online administration		*
ability to reserve space for a file and set fixed extent sizes and allocation flags		*
online snapshot file system for backup		*
direct I/O, supporting improved database performance		*
data synchronous I/O		*
DMAPI (Data Management API)		*

How to tell if JFS 3.5 is installed

To determine if a `vmunix` file has JFS3.5 compiled into it, you can run:

```
what /stand/vmunix | grep libvxfs.a
```

or

```
nm /stand/vmunix | grep vx_work
```

If you get output from either of these two commands, then the `vmunix` file has JFS 3.5 compiled into it, e.g.:

Module 10

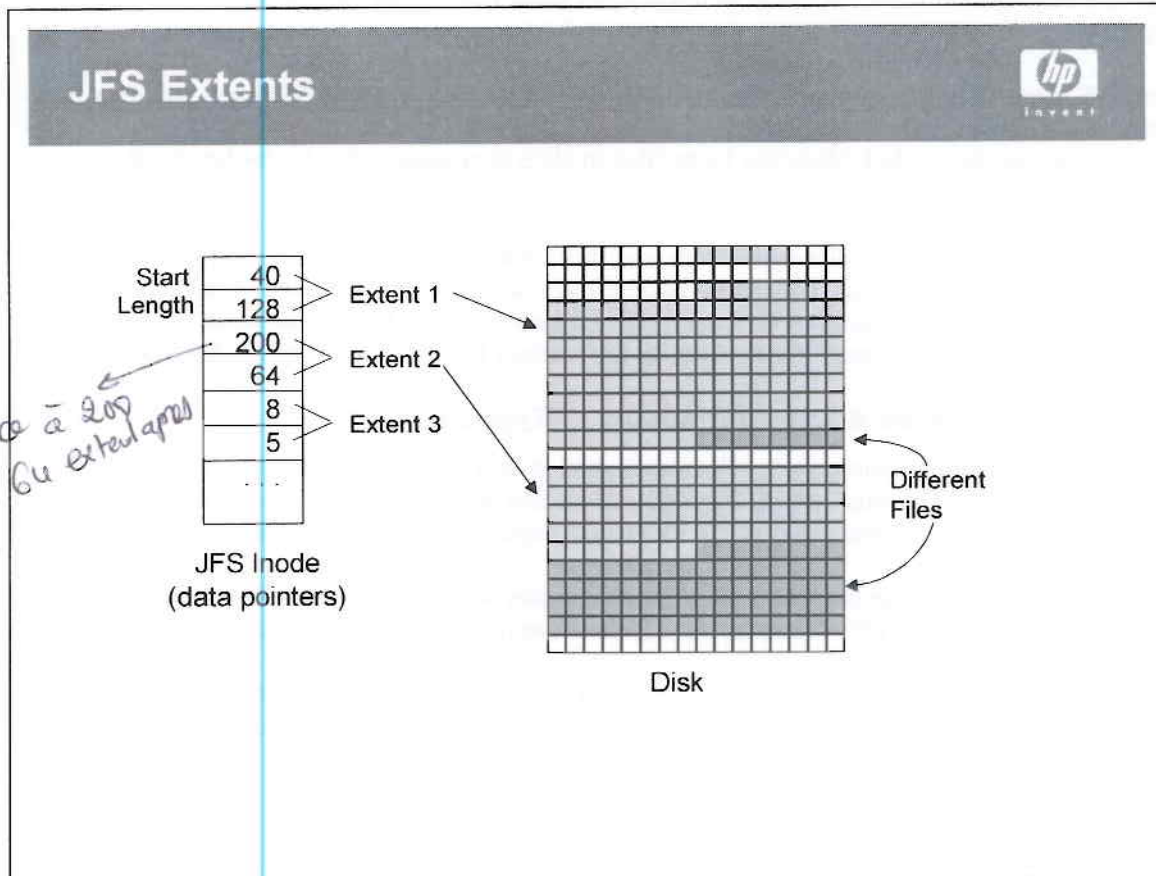
VxFS Performance Issues

```
# what /stand/vmunix | grep libvxfs.a
$Revision: libvxfs.a: CUPI80_BL2000_1108_2 Wed Nov 8 10:59:
22 PST 2000 $

# nm /stand/vmunix | grep vx_work
[13585] | 9746968 | 8|OBJT |LOCAL|0| .rodata|$S704$vx_workli
st_gettag
[13587] | 9746976 | 8|OBJT |LOCAL|0| .rodata|$S705$vx_workli
st_enqueue
[13589] | 9746984 | 8|OBJT |LOCAL|0| .rodata|$S706$vx_workli
st_thread
[13591] | 9746992 | 8|OBJT |LOCAL|0| .rodata|$S707$vx_workli
st_process
[13593] | 9747000 | 8|OBJT |LOCAL|0| .rodata|$S708$vx_workth
read_set
[34118] | 991200 | 232|FUNC |GLOB |0| .text|vx_worklist_enq
ueue
[27664] | 1940288 | 96|FUNC |GLOB |0| .text|vx_worklist_get
tag
[23820] | 13229528 | 40|OBJT |GLOB |0| .bss|vx_worklist_hig
h
[22805] | 13182888 | 16|OBJT |GLOB |0| .bss|vx_worklist_lk
[36804] | 13762256 | 40|OBJT |GLOB |0| .bss|vx_worklist_low

[39078] | 1744792 | 436|FUNC |GLOB |0| .text|vx_worklist_pro
cess
[33997] | 1745344 | 196|FUNC |GLOB |0| .text|vx_worklist_thr
ead
[23090] | 12350056 | 8|OBJT |GLOB |0| .sbss|vx_worklist_thr
ead_sv
[36954] | 1745232 | 84|FUNC |GLOB |0| .text|vx_worklist_wak
eup
[31238] | 2034928 | 48|FUNC |GLOB |0| .text|vx_workthread_c
reate
[13579] | 7215680 | 232|FUNC |LOCAL|0| .text|vx_workthread_s
et
```


10-3. SLIDE: JFS Extents



Student Notes

JFS allocates space to files in the form of **extents** - adjacent blocks of disk space treated as a unit. Extents can vary in size from a single block (minimum 1 KB in size) to many megabytes. Organizing file storage in this manner allows JFS to better support large I/O requests, with more efficient reading and writing to continuous disk space areas.

JFS extents are represented by a starting block number and a block count. In the example on the slide, the first extent starts at block 40 and contains a length of 128 blocks (or 128 KB, assuming blocks are 1KB in size). When the file grew past the 128 KB size, JFS tried to increase the size of the last extent. Since another file was already occupying this location, a new extent was allocated, starting at block 200. This extent grew to a size of 64 KB, before encountering another file. At this point, a third extent was allocated at block 8. Initially, 8 KB were allocated to the third extent, but upon closing the file, any space not used by the last extent is returned to the operating system. Since only 5 KB were used, the extra 3 KB were returned.

Direct and Indirect Extents in Version 2 Disk Layout

Unlike the HFS inode, the vxfs inode is 184 (rather than 128) bytes long and contains direct and indirect pointers.

In the HFS inode, the pointers address data blocks (8K by default) with 12 direct pointers and 3 additional indirect pointers for single, double and triple indirection. In reality triple indirection is rarely needed. Mapping large files in HFS is complex due to the levels of indirection needed to address many 8K blocks.

The JFS (vxfs) inode has 10 direct pointers and three additional pointers for single, double, and triple indirect addressing. The pointers no longer address single blocks of data but rather large extents of data. It is unlikely that any indirect pointers will be needed at all as the 10 direct pointers can define large spaces due to the variant length of the extents themselves.

Version 3 and Version 4 Extent Mapping.. “Typed Extents”.

The above discussion is true only for Version 2 disk layout. In addition to the above, in V3/V4 we also have “Typed Extents” which basically allow any level of indirection allowing very large files to be created from many small extents if required (this is not desirable however!).

Version 2 also imposes the limit that all indirect extents be the same size (direct extents can be variable in length). In V3/V4 we can have indirect extents of any size mix.

V3/V4 will always attempt to use the simplest approach. We use the 10 direct pointers when we can! Inodes are converted to Typed Indirect when the file exceeds the capability of 10 direct extents.

10-4. SLIDE: Extent Allocation Policies

Extent Allocation Policies



- Disk space allocation: the Block Size can be 1K, 2K, 4K, 8K.
- Extents are predefined in free space - "Power of 2 Rule"
- Preferred allocation rules
- Largest single extent is 16MB (with 8K block size).
- Full use of Single Indirection in default HFS would also be 16MB
- VxFS supports large files without indirection.

Student Notes

Disk Space Allocation: The Block Size

Disk space is allocated by the system in 1024 byte device blocks (**DEV_BSIZE**). An integral number of device blocks are grouped together to form a file system block. VxFS supports file system block sizes of 1024, 2048, 4096, and 8192 bytes.

The default block size is:

- 1024 bytes for file systems less than 8 gigabytes;
- 2048 bytes for file systems less than 16 gigabytes;
- 4096 bytes for file systems less than 32 gigabytes;
- 8192 bytes for file systems 32 gigabytes or larger.

The block size may be specified as an argument to the **mkfs** or **newfs** utility and may vary between VxFS file systems mounted on the same system. VxFS allocates disk space to files in extents. An extent is a set of contiguous blocks (up to 2048 blocks in size).

Extents in Free Space - "Power of 2 Rule"

Free space is described by bitmaps in each allocation unit. The allocation units are split into 16 "sections". Each section has a series of bitmaps that represent all the possible extents with sizes from 1 block to 2048 blocks by powers of 2. The first bitmap represents all the blocks in the section as one block extents, the second as two block extents, the third as four block extents, etc.

The first bitmap, of 2048 bits, represents the section as 2048 one-block extents. The second bitmap, of 1024 bits, represents the section as 1024 two-block extents. This continues for all powers of 2 up to the single bit that represents one 2048 block extent. The file system uses this bitmapping scheme to find an available extent closest in size to the space required. This keeps files as contiguous as possible for faster performance.

The largest possible extent on a file in a VxFS file system (with the largest block size of 8 KB) is $2048 * 8 \text{ KB} = 16 \text{ MB}$.

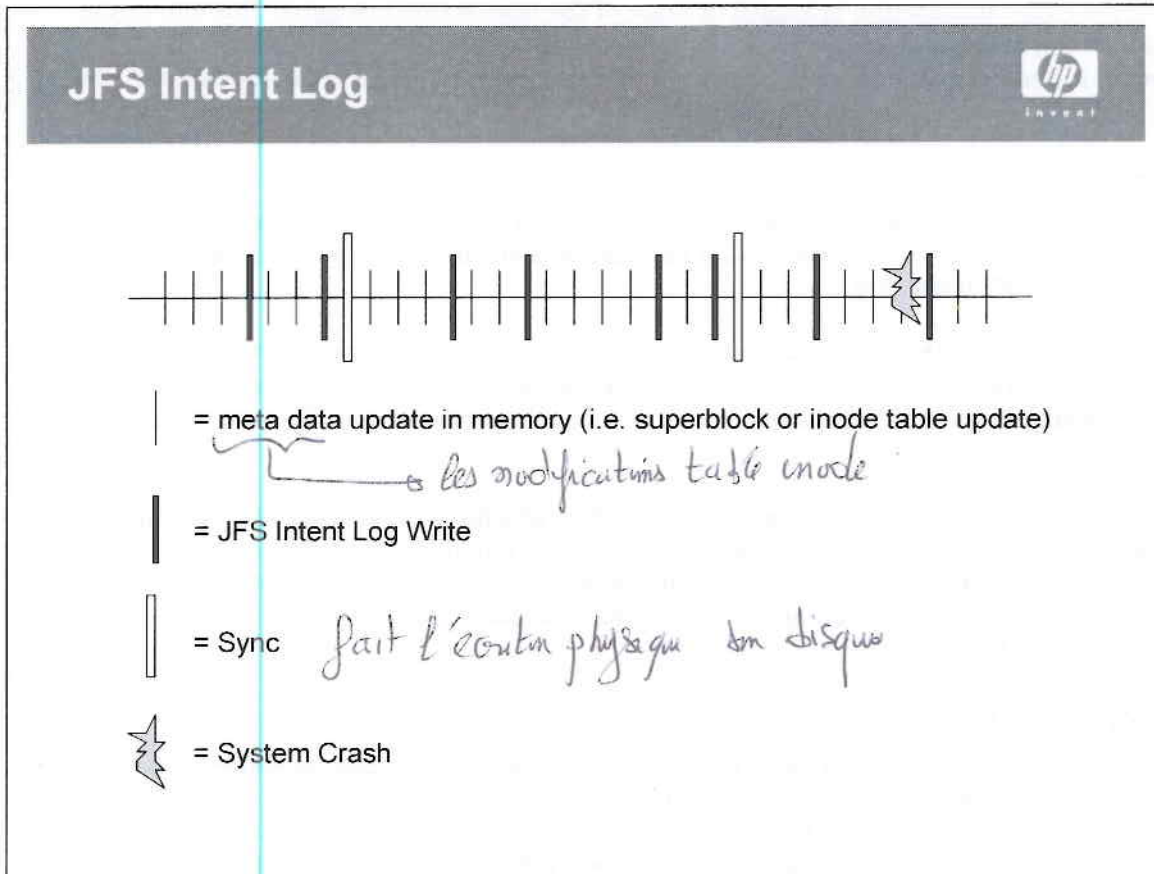
Preferred Allocation

The following rules are satisfied wherever possible starting with the preferred rules at the top and working down to less preferred rules.

- Allocate files using contiguous *extent* of blocks
- Attempt to allocate each file in one extent of blocks
- If not possible, attempt to allocate all extents for a file close to each other
- If possible, attempt to allocate all extents for a file in the same *allocation unit*

An allocation unit is an amount of contiguous (and therefore close together) file system space equal to 32 MB in size. It is roughly analogous to the HFS cylinder group, but is not dependent on the geometry of the disk drive in any way.

10-5. SLIDE: JFS Intent Log



Student Notes

A key advantage of JFS is that all file system transactions are written to an Intent Log. The logging of file system transactions helps to ensure the integrity of the file system, and allows the file system to be recovered quickly in the event of a system crash.

How the Intent Log Works

When a change is made to a file within the file system, such as a new file being created, a file being deleted, or a file being updated, a number of updates must be made to the superblock, inode table, bit maps, and other structures for that file system. These changes are called **metadata** updates. Typically, there are multiple metadata updates, which take place every time a change is made to a file.

With JFS, after every successful file change (also called a transaction), all the metadata updates related to that transaction get written out to a JFS Intent Log. The purpose of the Intent Log is to hold all *completed* transactions that have not yet been flushed out to disk.

If the system were to crash, the file system could quickly be recovered by checking the file system and applying all transactions in the intent log. Since only completed transactions are logged, there is no risk of a file change being only partially updated (i.e. only some metadata

updates related to the transactions being logged, and other metadata updates related to the same transaction not being logged). The logging of only COMPLETED transactions prevents the file system from being out-of-sync due a crash occurring in the middle of a transaction. Either the entire transaction is logged or none of the transaction is logged. This allows the JFS intent log to be used in a recovery situation as opposed to a standard **fsck**. The JFS recovery is done in seconds, as opposed to a standard **fsck** that (on a big file system) could take minutes, or even hours.

Example

Using the example on the slide, assume that each file transaction requires from one to four metadata updates. After each successful file transaction, all the related metadata updates are written to the JFS intent log.

After 30 seconds, all the metadata updates are written out to disk by the **sync** daemon, and a corresponding DONE record is written to the JFS intent log for each JFS transaction that was flushed during the **sync**. The system can now reuse that space in the JFS intent log for new JFS transactions.

When a crash occurs (in our example, in the middle of a file transaction), the uncompleted transaction never has any metadata written to the JFS intent log; therefore only one transaction is in the JFS intent log since the last **sync**. Only this transaction needs to be redone and then the file system is recovered and in a stable state. Compare this with having to do a standard **fsck**.

Performance Impacts

The intent log size is chosen when a file system is created and cannot be subsequently changed. The **mkfs** utility uses a default intent log size of 1024 blocks. The default size is sufficient for most workloads. If the system is used as an NFS server, for intensive synchronous write workloads, or for dynamic "work file loads" with many metadata changes, performance may be improved using a larger log size.

File data is not normally written to the intent log. However, if the application has designated to do synchronous writes and the writes are 32 KB or smaller, the file data *will* be written to the intent log, along with the meta-data. This behavior can be modified by mount options (discussed later in this module).

With larger intent log sizes, recovery time is proportionately longer and the file system may consume more system resources (such as memory) during normal operation.

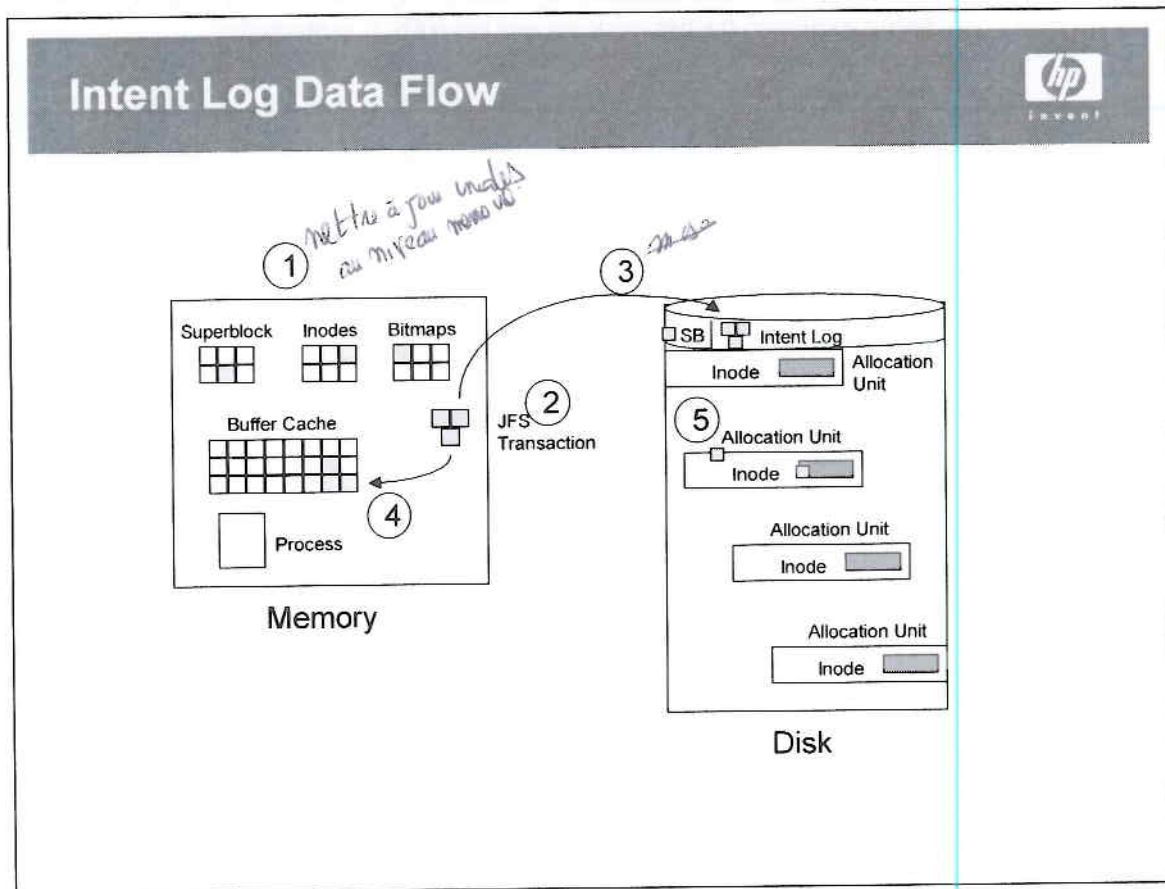
There are several system performance benchmark suites for which VxFS performs better with larger log sizes. As with block sizes, the best way to pick the log size is to try representative system loads against various sizes and pick the fastest.

The performance degradation occurs when the entire JFS intent log becomes filled with pending JFS transactions. In these situations, all new JFS transactions must wait for DONE records to arrive for the existing JFS transactions. Once the DONE records arrive, the space used by the corresponding transactions can be freed and reused for new transactions.

Having to wait for DONE records to arrive can significantly decrease performance with JFS. In these cases, it is suggested the JFS file system be reinitialized with a larger JFS intent log.

<i>CAUTION:</i>	Network file systems (NFS) can generate a large number of metadata updates if accessed currently by multiple systems. For JFS file systems being exported for network access via NFS, it is strongly recommended these file systems have an intent log size of 16 MB (maximum size for intent log).
-----------------	---

10-6. SLIDE: Intent Log Data Flow



Student Notes

The following slide shows a graphical representation of how JFS transactions are processed.

System call is issued (for example, write call).

1. All in-memory data structures related to the transaction are updated. These in-memory structures would include the superblock, the inode table, and the bitmaps.
2. Once the in-memory structures are updated, a JFS transaction is packaged containing the modifications to the in-memory structures. This packaged transaction contains all the data needed to reproduce the transaction (should that be necessary).
3. Once the JFS transaction is created, it is written to the intent log. (When it is written depends on mount options.)

At this point, control is returned to the system call.

4. Since the transaction is now stored on disk (in the intent log), there is no hurry to flush the in-memory data structures to their corresponding disk-based data structures.

Therefore, the in-memory structures are transferred to the buffer cache, and the **sync** daemon flushes out these transactions within the next 30 seconds.

5. After the metadata structures are flushed out, a DONE record is written to the intent log indicating the transaction has been updated to disk, and the corresponding transaction no longer needs to be kept in the intent log.

10-7. SLIDE: Understand Your I/O Workload

Understand your I/O Workload



- Is it data-intensive?
 - few files, large chunks being shuffled around
- Is it attribute-intensive?
 - many files, small chunks being shuffled
- Is the access pattern random or sequential I/O?
 - Check for `read()`, `write()`, and `lseek()` system calls
- What is the bandwidth and size of the I/Os?
 - Are these consistent?
 - Spindles Win Prizes! LVM or VxVM Stripes
 - Use XP Disk Arrays

Student Notes

Understand your I/O Workload

Tuning the file system's parameters to optimize performance can only be done effectively when you know what type of I/Os the application is doing. It would be wrong to tune for large block size and maximum contiguous space allocation if the application does many small random I/Os to many small files.

Data Intensive?

Commercial data base applications generally deal with very large files in the table space and large I/Os to those files. Any high degree of small random I/O should be taken care of by the data base's own buffers (System Global Area) and the HP-UX buffer cache (if it is being used).

We may choose to increase the block size in this situation and tune for maximum read ahead/write behind. The following slides will cover this type of tuning.

Attribute Intensive?

Some applications generate many small I/Os to many small files. In this situation a large block size and maximum read ahead/write behind would be inappropriate, generating more I/O than is necessary. A Mail Server or Web Server could be regarded as such an application.

Sequential or Random IO?

We need to characterize the I/O from an application into sequential or random types. Again, in general sequential I/O will benefit from larger block size and continuous files and random I/O will require smaller block size to increase the number of blocks that can be maintained in the buffer cache.

With sequential I/O we are more interested in maximizing the MB/sec throughput of the disk (as seen with **sar** or **glance** etc). With random I/O we will be looking to the "I/Os Per Second" metrics associated with the disk (r+w/s in **sar**). Remember that the fastest random I/Os we do are the ones that never go to the disk (!) because they are in the buffer cache (we hope).

The Direct I/O feature of OnlineJFS 3.5 is an attempt to recognize when I/Os to a file are very large and sequential. Direct I/O will then attempt to bypass the buffer cache to benefit the generator of the large I/Os in question. Most applications are not designed to handle their own buffering and will lose a great deal of performance if they attempt to use Direct I/O.

Disk Bandwidth

In the end we can only get so much performance out of a single spindle. Modern fast disks (10,000+ RPM, 5ms access time) can only provide an absolute maximum of approx 10 MB/s for very sequential I/O and around 150 I/Os Per Second. Once your file system is extracting these sorts of numbers (or even 50% of them!) you can consider that the hardware has become the limiting factor. Stop tuning and buy more disks!

Remember that "spindles win prizes". LVM or VxVM striping will help in this situation as the single spindle performance can be aggregated by the number of spindles.

Using expensive RAID technology like the HP XP256, XP512, or XP1024 Disk Arrays will also improve apparent "spindle" performance. The author has seen a single XP512 logical device provide a sustained 60MB/s read performance for sequential I/O and over 1500 I/Os per second for a single threaded random application test to a single logical device.

10-8. SLIDE: Performance Parameters

Performance Parameters



Things that an administrator can change to optimize JFS:

- Choosing a Block Size
- Choosing an Intent Log Size
- Choosing Mount Options
- Kernel Tunables
 - Internal Inode Table Size
- Monitoring Free Space and Fragmentation
- Changing extent attributes on individual files
- I/O Tuning
 - Tunable VxFS I/O Parameters
 - Command Line
 - Configuration file (`/etc/vx/tunefstab`)

Student Notes

We will discuss the following choices over the next slides. Note that the some parameters can only be set when the file system is created.

- **At file system creation time (only):**
 - Choosing a Block Size
 - Choosing an Intent Log Size
- **After file system creation:**
 - Choosing Mount Options
 - Kernel Tunables
 - Kernel Inode Table Size
 - Monitoring Free Space and Fragmentation
 - Changing extent attributes on individual files
 - I/O Tuning
 - Tunable VxFS I/O Parameters

10-9. SLIDE: Choosing a Block Size

Choosing a Block Size



- Choose the right block size for the application.
- Consider maximum block size (8K) for large file data base
 - Small files will waste space
 - System overhead will be less
 - Files approaching 1GB are “large”
- Consider minimum block size (1K) for small file mail server or web server
 - More system overhead if files are large
- Use large block size for sequential I/O application
- Use small block size for random I/O application

Student Notes

You specify the block size when a file system is created; it cannot be changed later. The standard HFS file system defaults to a block size of 8K with a 1K fragment size. This means that space is allocated to small files (up to 12 blocks) in 1K increments. Allocations for larger files are done in 8K increments. Because many files are small, the fragment facility saves a large amount of space compared to allocating space 8K at a time.

The unit of allocation in VxFS is a block. There are no fragments because storage is allocated in extents that consist of one or more blocks. The smallest block size available is 1K, which is also the default block size for VxFS file systems created on devices of less than 8 gigabytes.

Choose a block size based on the type of application being run. For example, if there are many small files, a 1K block size may save space.

For large file systems, with relatively few files, a larger block size is more appropriate. The trade-offs of specifying larger block sizes are: 1) a decrease in the amount of space used to hold the free extent bitmaps for each allocation unit, 2) an increase in the maximum extent size, and 3) a decrease in the number of extents used per file versus an increase in the amount of space wasted at the end of files that are not a multiple of the block size.

Module 10

VxFS Performance Issues

Larger block sizes use less disk space in file system overhead, but consume more space for files that are not a multiple of the block size. The easiest way to judge which block sizes provide the greatest system efficiency is to try representative system loads against various sizes and pick the fastest.

Specifying the Block Size

The following **newfs** command creates a VxFS file system with the maximum block size and support for large files.

```
# newfs -F vxfs -b 8192 -o largefiles /dev/vgjfs/rlvol1
```

The block size for files on the file system represents the smallest amount of disk space that can be allocated to a file. It must be a power of 2 selected from the range 1024 to 8192. The default is 1024 for file systems less than 8 gigabytes, 2048 for file systems less than 16 gigabytes, 4096 for file systems less than 32 gigabytes, and 8192 for larger file systems.

10-10. SLIDE: Choosing an Intent Log Size

Choosing an Intent Log Size



- Intent log size cannot be changed after file system creation
- `mkfs` applies a default log size of 1024 blocks
- Performance may improve as using a larger log size
 - NFS server will benefit from a 16MB (largest) log size
 - Synchronous write intensive applications

actuellement 32MB

Student Notes

The intent log size is chosen when a file system is created and cannot be changed afterwards.

The default intent log size chosen by `mkfs` is 1024 blocks and is suitable in most situations.

For some types of applications (NFS server or intensive synchronous write loads), performance may be improved by increasing the size of the intent log.

Note that recovery time will also be proportionally longer as the log size increases.

Memory requirements for the log maintenance will also increase as the log size increases. Ensure that the log size is not more than 50% of the physical memory size of the system or `fsck` will not be able to fix it after a system crash.

Ideal log size for NFS is 2048 with a file system block size of 8192.

Specifying the Intent Log Size

To create a VxFS file system with a default block size and a 16MB intent log:

```
# newfs -F vxfs -o logsize=16384 /dev/vgjfs/rlvol1
```

“-o logsize=” specifies the number of file system blocks to allocate for the transaction-logging area. It must be in the range of 32 to 16384 blocks.

The minimum number for Version 2 disk layouts is 32 blocks.

The minimum number for Version 3 and Version 4 disk layouts is the number of blocks that make the log no less than 256K.

If the file system is:

- greater than or equal to 8MB, default is 1024 blocks
- greater than or equal to 2MB, and less than 8 MB, default is 128 blocks
- less than 2MB, default is 32 blocks

While `logsize` is specified in blocks, the maximum size of the intent log is 16384 KB. This means the maximum values for `logsize` are:

- 16384 for a block size of 1024 bytes
- 8192 for a block size of 2048 bytes
- 4096 for a block size of 4096 bytes
- 2048 for a block size of 8192 bytes

10-11. SLIDE: Intent Log Mount Options

Intent Log Mount Options



- Full logging*
- Delayed logging
- Temporary logging
- No logging**
- Disallow small sync I/Os in log
- Force clear new file blocks

log : je vas tous tracer en mode synchron

delaylog : de base synchron mais c'est la machine trame il pasent en delay

tmplog →

nolog →

nodatainlog (50% perf cost!)

blkclear (10% perf cost!)

*Note only the first option is default for mount

on ne jamais de No logging car si crash la machine
va travailler du heure pour fsck data

Student Notes

JFS offers mount options to delay or disable transaction logging to the intent log. This allows the system administrator to make trade-offs between file system integrity and performance.

Following are the logging options:

Mount Option	Description
Full logging (log)	File system structural changes are logged to disk before the system call returns to the application (synchronously). If the system crashes, fsck (1M) will complete logged operations that have not completed.
Delayed logging (delaylog)	Some system calls return before the intent log is written. This improves the performance of the system, but some changes are not guaranteed until a short time later when the intent log is written. This mode approximates traditional UNIX system guarantees for correctness in case of system failure.
Temporary logging	The intent log is almost always delayed. This improves

Module 10
VxFS Performance Issues

(tmplog)

performance, but recent changes may disappear if the system crashes. This mode is only recommended for temporary file systems.

No logging (nolog)

The intent log is disabled. The other three logging modes provide for fast file system recovery; **nolog** does not provide fast file system recovery. With **nolog** mode, a full structural check must be performed after a crash. This may result in loss of substantial portions of the file system, depending upon activity at the time of the crash. Usually, a **nolog** file system should be rebuilt with **mkfs (1M)** after a crash. The **nolog** mode should only be used for memory resident or very temporary file systems.

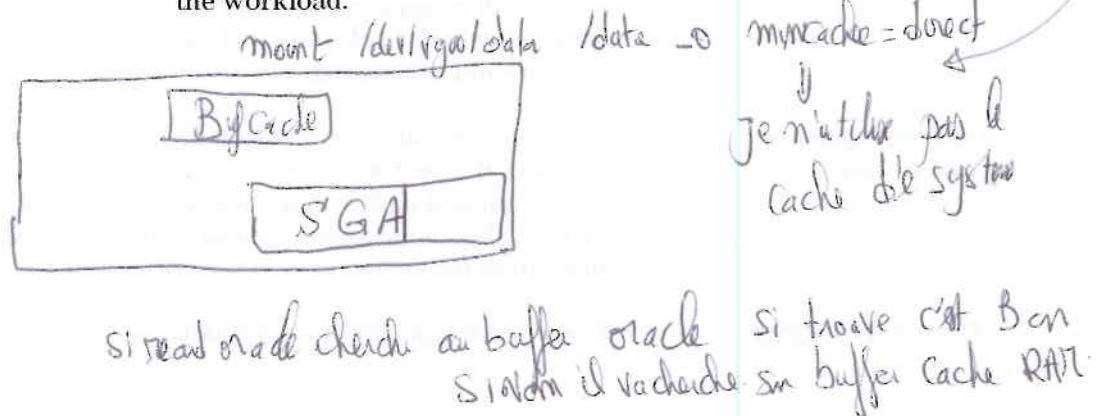
nodatainlog

The **nodatainlog** mode should be used on systems with disks that do revectoring. Normally, a VxFS file system uses the intent log for synchronous writes. The inode update and the data are both logged in the transaction, so a synchronous write only requires one disk write instead of two. When the synchronous write returns to the application, the file system has told the application that the data is already written. If a disk error causes the data update to fail, then the file must be marked bad and the entire file is lost. If a disk supports bad block revectoring, then a failure on the data update is unlikely, so logging synchronous writes should be allowed. If the disk does not support bad block revectoring, then a failure is more likely, so the **nodatainlog** mode should be used.

A **nodatainlog** mode file system should be approximately 50 percent slower than a standard mode VxFS file system for synchronous writes. Other operations are not affected.

blkclear

The **blkclear** mode is used in increased data security environments. The **blkclear** mode guarantees that uninitialized storage never appears in files. The increased integrity is provided by clearing extents on disk when they are allocated to a file. Extending writes are not affected by this mode. A **blkclear** mode file system should be approximately 10 percent slower than a standard mode VxFS file system, depending on the workload.



10-12. SLIDE: Other JFS Mount Options

Other JFS Mount Options (data i/o)



mincache options (buffer cache)	convosync options (synchronous I/O)
• closesync* • direct • dsync • unbuffered • tmpcache	• closesync • direct <i>c'est les plus recommander pour Oracle</i> • dsync • unbuffered • delay

Laisse la machine asynchron c'est un peu plus rapide

* NOTE: This is the only additional option available with BaseJFS, all other options require OnlineJFS.

mount /

Student Notes

Understanding asynchronous, data synchronous (O_DSYNC) and fully synchronous (O_SYNC) application I/O.

When an application program opens a file with the `open()` system call, the programmer makes a decision on how the I/Os will occur between the application memory and the file system. The following three options are available, in order, ranging from highest performance (lowest integrity) to lowest performance (best integrity). In this discussion "integrity" refers to the potential damage to file system structures and customer data during a system crash.

1. Asynchronous I/O Standard Mode High performance / Low integrity

In asynchronous mode, all application I/Os are done to buffer cache including data and inode modifications. The `write()` system call will return quickly to the application which can continue "in faith" that the data will make it to the disk. Data integrity will be fully compromised by a system crash and new "just created" files may even disappear.

2. **Data Synchronous I/O** **O_DSYNC** Low performance / Good integrity

If the file is opened with the **O_DSYNC** flag, the file is in "Data Synchronous" mode. In this situation, **write()** system calls that modify data do not return until the disk has acknowledged the receipt of the data. However, some inode changes (time stamps, etc.) are still performed asynchronously and may not have arrived at the disk in the case of a system crash.

3. **Synchronous I/O** **O_SYNC** Lowest performance / Best integrity

Fully synchronous behavior is obtained by opening the file with **O_SYNC**. All operations are now synchronous and **write()** system calls block for both data and inode modifications. Minimal damage will now occur in the event of a system crash.

mincache vs. convosync

mincache manipulates the behavior of the buffer cache. All of the **mincache** options except **mincache=closesync** require the OnlineJFS product (see slide).

convosync ("convert osync") changes the behavior of data synchronous (**O_DSYNC**) and synchronous (**O_SYNC**) writes. All **convosync** options require OnlineJFS.

The **mincache** and **convosync** options generally control the integrity of the user data, where the log options (**log**, **delaylog**, **tmplog**, **nolog**) control the integrity of the **metadata** only.

mincache

mincache=closesync Flush data to disk synchronously when file is closed.

The **mincache=closesync** mode is useful in desktop environments where users are likely to shut off the power on the machine without halting it first. In this mode, any changes to the file are flushed to disk synchronously when the file is closed.

To improve performance, most file systems do not synchronously update data and inode changes to disk. If the system crashes, files that have been updated within the past minute are in danger of losing data. With the **mincache=closesync** mode, if the system crashes or is switched off, only files that are currently open can lose data. A **mincache=closesync** mode file system should be approximately 15 percent slower than a standard mode VxFS file system, depending on the workload.

mincache=direct Bypass the buffer cache for all data and inode changes, forces fully synchronous behavior and totally skips buffer cache.

mincache=unbuffered Bypass the buffer cache for data only. Inode changes are cached. Forces data synchronous-like behavior with no data in cache.

mincache=dsync

Equivalent to normal data synchronous behavior. Write does not return until data is on disk but data does go through buffer cache.

The `mincache=direct`, `mincache=unbuffered`, and `mincache=dsync` modes are used in environments where applications are experiencing reliability problems caused by the kernel buffering of I/O and delayed flushing of non-synchronous I/O.

The `mincache=direct` and `mincache=unbuffered` modes guarantee that all non-synchronous I/O requests to files will be handled as if the `VX_DIRECT` or `VX_UNBUFFERED` caching advisories had been specified. The `mincache=dsync` mode guarantees that all non-synchronous I/O requests to files will be handled as if the `VX_DSYNC` caching advisory had been specified. Refer to `vxfsio(7)` for explanations of `VX_DIRECT`, `VX_UNBUFFERED`, and `VX_DSYNC`.

The `mincache=direct`, `mincache=unbuffered`, and `mincache=dsync` modes also flush file data on close as `mincache=closesync` does.

mincache=tmpcache

Speeds up file growth by breaking data initialization rules.

The `-o mincache=tmpcache` option only affects *write extending* calls and is not available to files performing synchronous I/O. *write extending* calls refer to write calls that cause new file system blocks to be assigned to the file, extending the size of the file in blocks. The normal behavior for *write extending* calls is to write the new user data first, and insist on metadata to be written only *after* the user data.

Write extending calls are expensive from a performance standpoint, because the write call has to wait for the user data and the metadata to be written. A non-extending write call only requires the call to wait for the metadata.

With the `-o mincache=tmpcache` option, write extending calls do *not* have to wait for the user data to be written. This option allows the metadata to be written *before* user data (and the write call to return before the user data is written), significantly improving performance.

CAUTION:

The `-o mincache=tmpcache` option significantly increases the likelihood of non-initialized file system blocks (i.e. junk) appearing in files during a system crash. This is due to the file pointing to data blocks before the data is actually there. If the system crashes between the file's inode being updated (done first) and the user data being written (done second), then uninitialized data will appear in the file.

The `tmpcache` option should only be used for memory resident or very temporary file systems.

convosync

NOTE: Use of the **convosync=dsync** option violates POSIX guarantees for synchronous I/O.

The “convert osync” (**convosync**) mode has five values: **convosync=closesync**, **convosync=direct**, **convosync=dsync**, **convosync=unbuffered**, and **convosync=delay**.

The **convosync=closesync** mode converts synchronous and data synchronous writes to non-synchronous writes and flushes the changes in the file to disk when the file is closed.

The **convosync=delay** mode causes synchronous and data synchronous writes to be delayed rather than to take effect immediately. No special action is performed when closing a file. This option effectively cancels any data integrity guarantees normally provided by opening a file with **O_SYNC**. See **open(2)**, **fcntl(2)**, and **vxfsio(7)** for more information on **O_SYNC**.

Caution! Extreme care should be taken when using the **convosync=closesync** or **convosync=delay** mode because they actually change synchronous I/O into non-synchronous I/O. This may cause applications that use synchronous I/O for data reliability to fail, if the system crashes and synchronously written data is lost.

The **convosync=direct** and **convosync=unbuffered** mode convert synchronous and data synchronous reads and writes to direct reads and writes, bypassing the buffer cache.

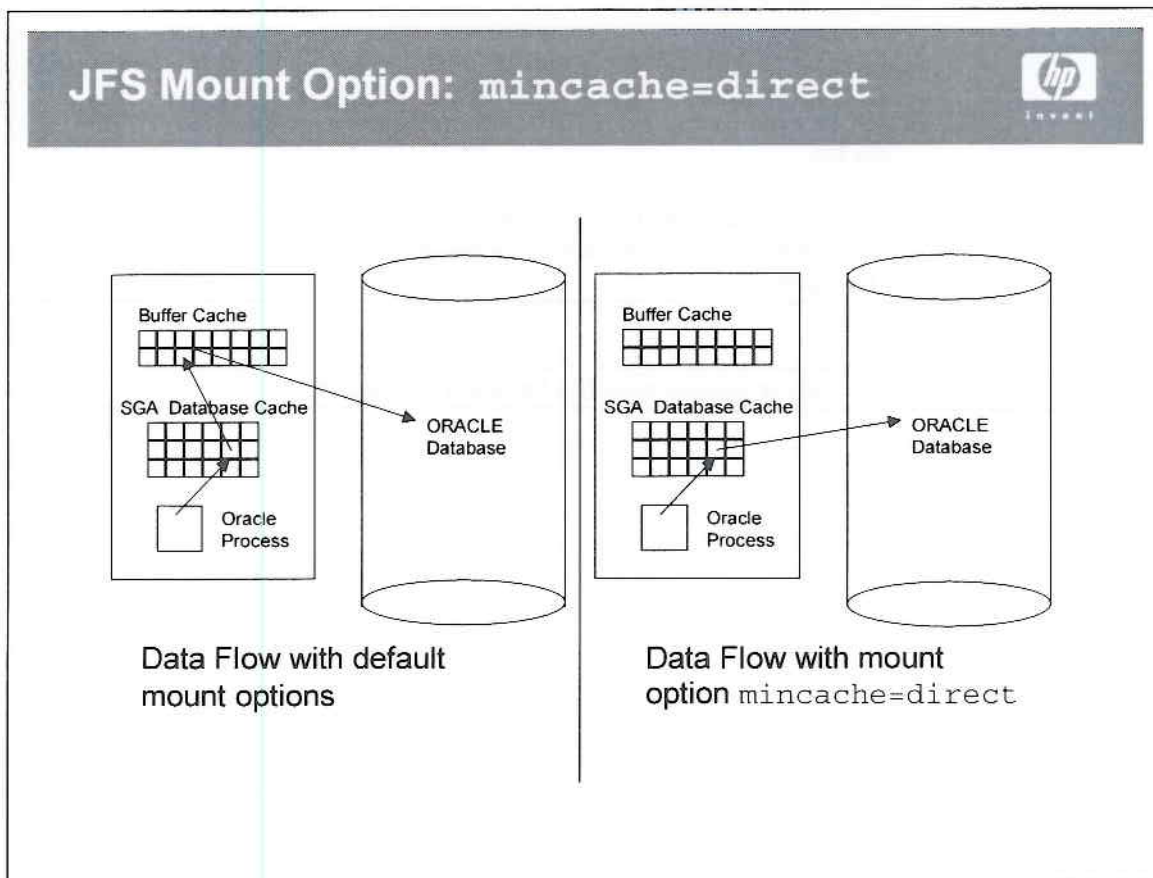
The **convosync=dsync** mode converts synchronous writes to data synchronous writes.

As with **closesync**, the **direct**, **unbuffered**, and **dsync** modes flush changes in the file to disk when it is closed. These modes can be used to speed up applications that use synchronous I/O. Many applications that are concerned with data integrity specify **O_SYNC** in order to write the file data synchronously. However, this has the undesirable side effect of updating inode times and therefore slowing down performance.

The **convosync=dsync**, **convosync=unbuffered**, and **convosync=direct** modes alleviate this problem by allowing applications to take advantage of synchronous writes without modifying inode times as well.

NOTE: Before using **convosync=dsync**, **convosync=unbuffered**, or **convosync=direct**, make sure that all applications that use the file system do not require synchronous inode time updates for **O_SYNC** writes.

10-13. SLIDE: JFS Mount Option: mincache=direct



Student Notes

The above slide illustrates the impact of setting the `-o mincache=direct` option.

By default, all JFS file system I/O goes through the system's buffer cache. When an application does its own caching (e.g. an Oracle database application), there are two levels of caching. One cache is managed by the application; the other cache is managed by the kernel. Using two caches is inefficient from both a performance and a memory usage standpoint (data exists in both caches).

When the file system is mounted with the `-o mincache=direct` option, it causes bypassing of the system's buffer cache and the data is written directly to disk. This improves performance and keeps the buffer cache available for other file systems that do not go through an application cache.

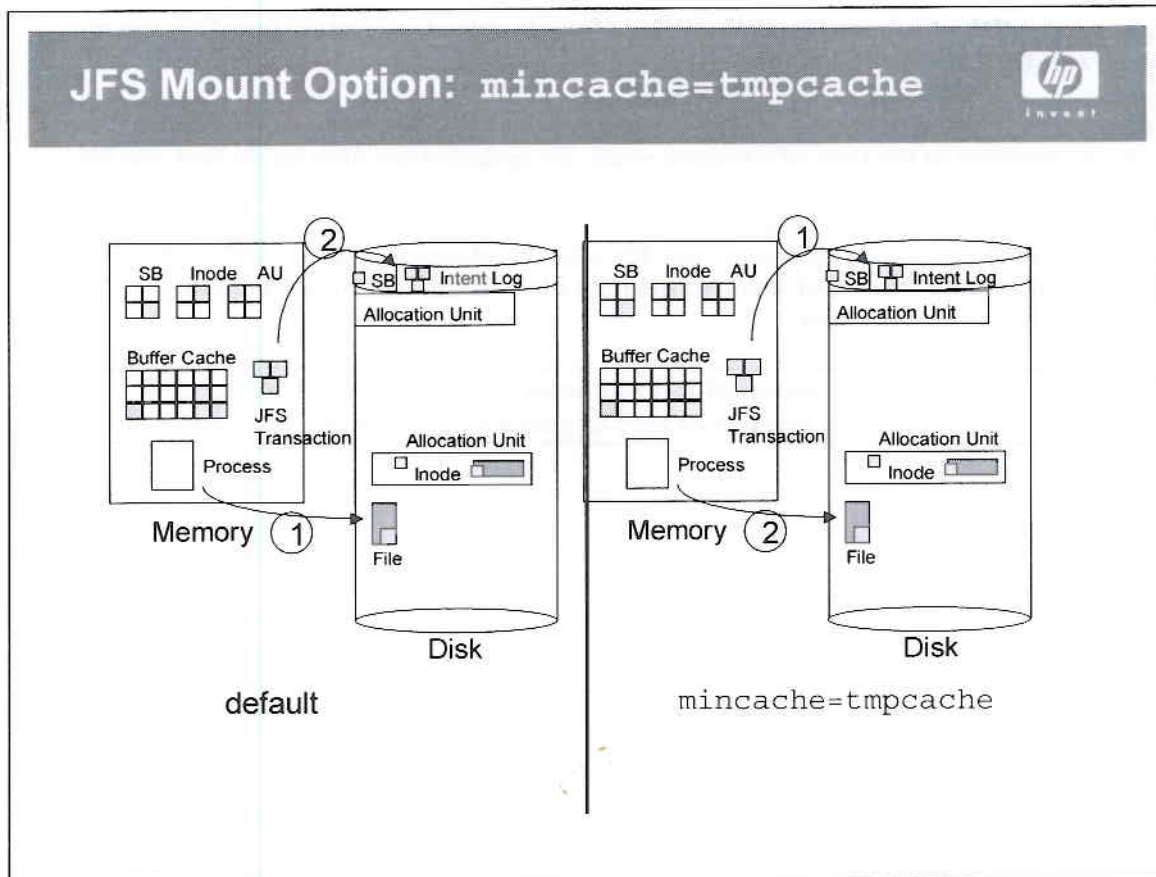
CAUTION: Use of the `-o mincache=direct` option can lead to a significant decrease in performance if used in the wrong situation. This option should only be used if:

1. An application creates and maintains its own data cache, *and*
2. All the files on the file system are cached in the application's data cache.

If there are some files being accessed on the mounted file system and these files are *not* being cached by the application, this option should not be used.

NOTE: This option is only available with the OnlineJFS product.

10-14. SLIDE: JFS Mount Option: mincache=tmpcache



Student Notes

By default, when a process performs a *write extending* call, the new data is written to disk before the file's inode is updated. In the slide above, the left side shows the **default** behavior:

1. Write data to newly allocated file system block.
2. Write JFS transaction meta-data out to the disk. The system call returns.

The advantage of this behavior is that uninitialized data will *not* be found within the file should a system crash occur. This is important from a data integrity standpoint.

The disadvantage of this behavior is slow performance, because the JFS transaction must wait for the user data I/O to complete before it can be written to the intent log.

Behavior with `-o mincache=tmpcache` Option

Performance can be improved (at the expense of data integrity) by mounting file systems with the `-o mincache=tmpcache` option. This option allows the JFS transactions to be written to the intent log before the user data is written to the file.

In the slide, the right side shows the **tmpcache** behavior:

Module 10
VxFS Performance Issues

1. Write JFS transaction out to disk. (The system call returns).
2. Write data to newly allocated file system block.

The advantage of this behavior is performance of *write extending* calls is fast. The system does not wait for the user data to be written to disk.

The disadvantage of this behavior is data integrity of the file is jeopardized, especially if the file is being updated at the time of a system crash. By updating the file's inode first, the file points to uninitialized data blocks which contains unknown data. The uninitialized file system blocks are expected to be initialized soon after the inode is updated; however, there still exists a small window of time when the file's inode references unknown data. If the system crashes during this small window, then the file will still be referencing the uninitialized data after the crash.

CAUTION:	The <code>-o mincache=tmpcache</code> option should only be used for memory resident or very temporary file systems.
-----------------	--

10-15. SLIDE: Kernel Tunables

Kernel Tunables



- VxFS inodes are cached in memory, separate from HFS.
- Kernel parameter `ninode` has no effect on VxFS.
- When `vx_ninode` is zero (default), inode cache is set in proportion to system memory (see table).
- `vx_ncsize` sets directory name lookup cache (1KB)

Student Notes

Internal Inode Table Size

VxFS caches inodes in an inode table (see Table below, "Inode Table Size"). There is a tunable in VxFS called `vx_ninode` that determines the number of entries in the inode table.

A VxFS file system obtains the value of `vx_ninode` from the system configuration file used for making the kernel (/stand/system for example). This value is used to determine the number of entries in the VxFS inode table. By default, `vx_ninode` is set to zero. The kernel then computes a value based on the system memory size.

Module 10
VxFS Performance Issues

Total Memory in Mbytes	Maximum Number of Inodes
8	400
16	1000
32	2500
64	6000
128	8000
256	16000
512	32000
1024	64000
2048	128000
8192	256000
32768	512000
131072	1024000

If the available memory is a value between two entries, the value of **vx_ninode** is interpolated.

Other VxFS Kernel Parameters

vx_ncsize Controls the size of the DNLC (directory name lookup cache) in the kernel. Recent directory path names are stored in memory to improve performance. This parameter is set in DNLC entries. The size of the DNLC is set to the sum of **ninode** and **vx_ncsize**.

10-16. SLIDE: Fragmentation

Fragmentation



- Keep file system free space over 10%
- Maintain free space distribution goals
 - Monitor with `df (1M)` or `fsadm (1M)`
- Repack files and free space with `fsadm -e`
 - Reduces the number of extents in large files
 - Makes small files contiguous (one extent)
 - Moves small recently used file closer to inode structures
 - Optimizes free space into larger extents
- Repack directories with `fsadm -d`
 - Remove empty entries from directories
 - Place recently used files at beginning of directory lists
 - Pack small directories directly in inode if possible

Student Notes

- **Keep file system free space over 10%**

In general, VxFS works best if the percentage of free space in the file system does not get below 10 percent. This is because file systems with 10 percent or more free space have less fragmentation and better extent allocation. Regular use of the `df(1M)` command to monitor free space is desirable. Full file systems should therefore have some files removed, or should be expanded (see `fsadm(1M)` for a description of online file system expansion).

- **Maintain free space distribution goals**

3 factors which can be used to determine the degree of fragmentation:

- percentage of free space in extents < than 8 blocks in length
- percentage of free space in extents < than 64 blocks in length
- percentage of free space in extents of 64 blocks or greater

An **unfragmented** file system will have the following characteristics:

- less than 1% of free space in extents < 8 blocks in length
- less than 5% of free space in extents < 64 blocks in length
- more than 5% of total file system size available as free extents in length of 64 or more blocks

A **fragmented** file system will have the following characteristics:

- greater than 5% of free space in extents < 8 blocks in length
- more than 50% of free space in extents < 64 blocks in length
- less than 5% of total file system size available as free extents in lengths of 64 or more blocks in size

Using df (1M)

The following example shows how to use **df** to map free space:

```
# df -F vxfs -o s /usr
/usr          (/dev/vg00/lvol7) :
Free Extents by Size
  1:      823    2:      206    4:      55    8:      206
 16:      158   32:      61   64:      48  128:      43
256:       23  512:      14 1024:       3 2048:       3
4096:        1 8192:        1 16384:       0 32768:       0
```

• Repack files and freespace

fsadm -e has the following goals for files and free data space

- Make “small” files (default: <64k) one contiguous extent
- Ensure that “large” files are built from large extents
- Move “small” and “recently used” (default: <14 days) files near inode area
- Move “large” or “old” (>14 days since last access) files to end of allocation unit
- Consolidate free space in center of data area

• Repack directories

fsadm -d has the following goals for directories

- Remove unused space from between used directory entries
- Pack directories and symbolic links into inode immediate area if possible
- Place directories and symbolic links first, then other files
- Sort each area by time of last access

fsadm (1M) Overview

Because blocks are allocated and deallocated as files are added, removed, expanded, and truncated, block space can become fragmented. This can make it more difficult for JFS to take advantage of the benefits provided by a contiguous extent allocation. To remove fragmentation, HP OnlineJFS includes a utility called **fsadm**, which will take fragmented blocks and reallocate them as contiguous extents. The **fsadm** utility can be run on a live file system (including one containing active databases) safely without interrupting data access.

The **fsadm** utility will bring the fragmented extents of files closer together, group them by type and frequency of access, and compact and sort directories. The **fsadm** utility is typically run as a recurring scheduled job and is an effective tool for the management of a high-performance online file system. Even if database software used on top of the file system has its own defragmenter, this additional defragmentation is necessary to make the storage that the database engine sees as contiguous as possible.

You can defragment (reorganize) your HP OnlineJFS file system using SAM or with **fsadm (1M)**, directly from the command line.

To use SAM:

1. Invoke SAM.
2. Select the **Disks and File Systems** functional area.
3. Select the **File Systems** application.
4. Select the JFS file system that you wish to reorganize from the directories' list.
5. Select the **Actions** menu.
6. Select the **VxFS Maintenance** menu item.
7. View reports on extent and directory fragmentation, then select **Reorganize Extents** or **Reorganize Directories** to defragment your JFS file system.

pour le faire on va créer un file system et tester
le **fsadm** (commande syntaxe)

fsadm -e -d

on peut le faire ~~par~~ temps (tranche temps)

10-17. TEXT PAGE: Monitoring and Repairing File Fragmentation

For optimal performance, the JFS extent allocator must be able to find large extents when it wants them. To maintain the file system performance levels, the **fsadm** utility should be run periodically against all JFS file systems to reduce fragmentation. The **fsadm** utility should be run between once a day and once a month against each file system. The frequency depends on file system usage and activity patterns and the importance of performance. The **-v** option can be used to examine the amount of work performed by **fsadm**. The frequency of reorganization can be adjusted, based on the rate of file system fragmentation.

To perform both directory and extent reorganization and to output reports on the directory and extent fragmentation before and after reorganization, enter the following:

```
# fsadm -F vxfs -d -D -e -E /<jfs_mount_point>
```

Reorganizing Options

- F vxfs** Specify the JFS file system type.
- D** Report on directory fragmentation. If specified in conjunction with the **-d** option, the fragmentation report is produced both before and after the directory reorganization.
- E** Report on extent fragmentation. If specified in conjunction with the **-e** option, the fragmentation report is produced both before and after the extent reorganization.
- d** Reorganize directories. Directory entries are reordered to place subdirectory entries first, then all other entries in decreasing order of time of last access. The directory is also compacted to remove free space.
- e** Extent reorganization. Attempt to minimize fragmentation. Aged files are moved to the end of the allocation units to produce free space. Other files are reorganized to have the minimum number of extents possible.
- s** Print a summary of activity at the end of each pass.
- v** Verbose. Report reorganization activity.
- a days** Consider files not accessed within the specified number of days as aged files. The default is 14 days. Aged files are moved to the end of the directory by the **-d** option and reorganized differently by the **-e** option.
- p passes** Maximum number of passes to run. The default is 5 passes. Reorganizations are processed until reorganization is complete or until the specified number of passes have been run.
- t time** Maximum time to run. Reorganizations are processed until reorganization is complete or the time limit has expired. *time* is specified in seconds.

If both the **-t** and **-p** options are specified, the utility exits if either of the terminating conditions is reached. If both the **-e** and **-d** options are specified, the utility will run all the directory reorganization passes before any extent reorganization passes.

fsadm uses the file **.fsadm** in the **lost+found** directory as a lock file. When **fsadm** is invoked, it opens the file **lost+found/.fsadm** in the root of the file system specified by **mount_point**. If the file does not exist, it is created. The **fcntl(2)** system call is used to obtain a write lock on the file. If the write lock fails, **fsadm** will assume that another **fsadm** is running and will fail. **fsadm** will report the process ID of the process holding the write lock on the **.fsadm** file.

Reporting on Directory Fragmentation

As files are allocated and freed, directories tend to grow and become sparse. In general, a directory is as large as the largest number of files it ever contained, even if some files have been subsequently removed.

The command line to obtain a directory fragmentation report is:

```
# fsadm -D /mountpoint_dir
```

The following is example output from the **fsadm -D** command:

```
# fsadm -D /home
```

Directory Fragmentation Report

		Dirs Searched	Total Blocks	Immed Dirs	Immeds to Add	Dirs to Reduce	Blocks to Reduce
au	0	15	3	12	0	0	0
au	1	0	0	0	0	0	0
total		15	3	12	0	0	0

The **Dirs Searched** column contains the total number of directories. A directory is associated with the extent-allocation unit containing the extent in which the directory's inode is located. The **Total Blocks** column contains the total number of blocks used by directory extents.

The **Immed Dirs** column contains the number of directories that are immediate, meaning that the directory data is in the inode itself as opposed to being in an extent. Immediate directories save space and speed path name resolution. The **Immeds to Add** column contains the number of directories that currently have a data extent, but that could be reduced in size and contained entirely in the inode.

The **Dirs to Reduce** column contains the number of directories for which one or more blocks can be freed, if the entries in the directory are compressed to make the free space in the directory contiguous. Because directory entries vary in length, large directories may contain a block or more of total free space, but with the entries arranged in such a way that the space cannot be made contiguous. As a result, it is possible to have a non-zero **Dirs to**

Reduce calculation immediately after running a directory reorganization. The **-v** (verbose) option of directory reorganization reports occurrences of failure to compress free space.

The **Blocks to Reduce** column contains the number of blocks that can be freed if the entries in the directory are compressed.

Measuring Directory Fragmentation

If the totals in the **Dirs to Reduce** column are substantial, a directory reorganization should improve the performance of path name resolution. The directories that fragment tend to be the directories with the most activity. A small number of fragmented directories may account for a large percentage of name lookups in the file system.

Directory Reorganization

If the **-d** option is specified, **fsadm** will reorganize the directories on the file system whose mount point is **mountpoint_dir**. Directories are reorganized in two ways: compressing and sorting.

For compression, the valid entries in the directory are moved to the front of the directory and the free space is grouped at the end of the directory. If there are no entries in the last block of the directory, the block is released and the directory size is reduced. If the directory entries are small enough, the directory is placed in the inode immediate data area.

The entries in a directory are also sorted to improve path name lookup performance. Entries are sorted based on the last access time of the entry. The **-a** option is used to specify a time interval; 14 days is the default if **-a** is not specified. The time interval is broken up into 128 buckets, and all times within the same bucket are considered equal. All access times older than the time interval are considered equal, and those entries are placed last. Subdirectory entries are placed at the front of the directory and symbolic links are placed after subdirectories, followed by the most recently accessed files.

The directory reorganization runs in one pass across the entire file system.

The command line to reorganize directories of a file system is:

```
fsadm -d [-s] [-v] [-p passes] [-t timeout] [-r rawdev] [-D] /mountpoint_dir
```

The following example illustrates the output of the command **fsadm -d -s** command:

```
# fsadm -d -s /home
```

Directory Reorganization Statistics

		Dirs Searched	Dirs Changed	Total Ioctl's	Failed Ioctl's	Blocks Reduced	Blocks Changed	Immeds Added
au	0	2343	1376	2927	1	209	3120	72
au	1	582	254	510	0	47	586	28
au	2	142	26	38	0	21	54	16
au	3	88	24	29	1	5	36	2
total		3155	1680	3504	2	282	3796	118

The **Dirs Searched** column contains the number of directories searched. Only directories with data extents are reorganized. Immediate directories are skipped. The **Dirs Changed** column contains the number of directories for which a change was made.

The **Total Ioctl's** column contains the total number of VX_DIRSORT **ioctl's** performed. Reorganization of directory extents is performed using this **ioctl**.

The **Failed Ioctl's** column contains the number of requests that failed. The reason for failure is usually that the directory being reorganized is active. A few failures should be no cause for alarm. If the **-v** option is used, all **ioctl** calls and status returns are recorded. The **Blocks Reduced** column contains the total number of directory blocks freed by compressing entries. The **Blocks Changed** column contains the total number of directory blocks updated while sorting and compressing entries.

The **Immeds Added** column contains the total number of directories with data extents that were compressed into immediate directories.

Reporting on Extent Fragmentation

As files are created and removed over time, the free extent map for an allocation unit will change from having one large free area to having many smaller free areas. This process is known as fragmentation. Also, when files are grown, particularly when growth occurs in small increments, small files can be allocated in multiple extents. In the ideal case, each file that is not sparse will have exactly one extent (containing the entire file), and the free-extent map is one continuous range of free blocks.

Conversely, in a case of extreme fragmentation, there can be free space in the file system, none of which can be allocated. For example, on Version 2 JFS file systems, the indirect-address extent size is always 8 KB long. This means that to allocate an indirect-address extent to a file, an 8-KB extent must be available. If no extent of 8 KB or larger is available, even though more than 8 KB of free space is available, an attempt to allocate a file into indirect extents will fail and return ENOSPC.

Determining Fragmentation

To determine whether fragmentation exists for a given file system, the free extents for that file system need to be examined. If a large number of small extents are free, there is fragmentation. If more than half of the amount of free space is taken up by small extents, (smaller than 64 blocks) or there is less than 5 percent of total file system space available in large extents, then there is serious fragmentation.

Running the Extent-Fragmentation Report

The extent-fragmentation report can be run to acquire detailed information about the degree of fragmentation in a given file system.

The following is the command line to run an extent-fragmentation report:

```
fsadm -E [-l largesize] /mountpoint_dir
```

The extent reorganizer has the concept of an immovable extent: if the file already contains large extents, reallocating and consolidating these extents will not improve performance, so

Module 10
VxFS Performance Issues

they are considered immovable. How large an extent must be to qualify as immovable can be controlled with the `-l` option. By default, *largsize* is 64 blocks, meaning that any extent larger than 64 blocks is considered to be immovable. For the purposes of the extent fragmentation report, the value chosen for *largsize* will affect which extents are reported as being immovable extents.

The following is an example of the output generated by the `fsadm -E` command:

```
# fsadm -E /home
```

Extent Fragmentation Report

		Files with	Total		Total
		Extents	Extents	Blocks	Distance
Au	0	14381	18607	30516	4440997
au	1	2822	3304	24562	927841
au	2	2247	2884	22023	1382962
au	3	605	780	24039	679867
total		19992	25575	101140	7431667

		Consolidatable		Immovable	
		Extents	Blocks	Extents	Blocks
au	0	928	2539	0	0
au	1	461	5225	99	13100
au	2	729	8781	58	11058
au	3	139	1463	49	17258
total		2257	18008	206	41416

Free Extents by Size

```

au 0 Free Blocks 217, Smaller Than 8 - 48%, Smaller Than 64 - 100%
    1:    15    2:    15    4:    15    8:    14
    16:    0   32:    0   64:    0  128:    0
    256:    0  512:    0 1024:    0 2048:    0
    4096:    0 8192:    0 16384:    0
au 1 Free Blocks 286, Smaller Than 8 - 41%, Smaller Than 64 - 100%
    1:    16    2:    21    4:    15    8:    13
    16:    4   32:    0   64:    0  128:    0
    256:    0  512:    0 1024:    0 2048:    0
    4096:    0 8192:    0 16384:    0
au 2 Free Blocks 510, Smaller Than 8 - 15%, Smaller Than 64 - 100%
    1:    10    2:    14    4:    10    8:    14
    16:    8   32:    6   64:    0  128:    0
    256:    0  512:    0 1024:    0 2048:    0
    4096:    0 8192:    0 16384:    0
au 3 Free Blocks 6235, Smaller Than 8 - 3%, Smaller Than 64 - 15%
    1:    29    2:    33    4:    27    8:    30
    16:   18   32:    8   64:    4  128:    3
    256:    2  512:    2 1024:    1 2048:    1
    4096:    0 8192:    0 16384:    0

```



```

au 4 Free Blocks 8551, Smaller Than 8 - 2%, Smaller Than 64 - 22%
  1:    29    2:    33    4:    30    8:    38
 16:    28   32:    29   64:    26   128:    11
256:     8   512:     3  1024:     0   2048:     0
4096:     0  8192:     0 16384:     0

```

```

total Free Blocks 15799, Smaller Than 8 - 4%, Smaller Than 64 - 24%
  1:    99    2:   116    4:    97    8:   109
 16:    58   32:    43   64:    30   128:    14
256:    10   512:     5  1024:     1   2048:     1
4096:     0  8192:     0 16384:     0

```

The numbers in the **Files with Extents** column indicate the total number of files that have data extents. A file is considered to be in the extent-allocation unit that contains the extent holding the file's inode.

The **Total Extents** column contains the total number of extents belonging to files in the allocation unit. The extents themselves are not necessarily in the same allocation unit.

The **Total Blocks** column contains the total number of blocks used by files in the allocation unit. If the total number of blocks is divided by the total number of extents, the resulting figure is the average extent size.

The **Total Distance** column contains the total distance between extents in the allocation unit. For example, if a file has two extents, the first containing blocks 100 through 107 and the second containing blocks 110 through 120, the distance between the extents is 110-107, or 3. In general, a lower number means that files are more contiguous. If an extent reorganization is run on a fragmented file system, the value for Total Distance should be reduced.

The **Consolidatable Extents** column contains the number of extents that are candidates to be consolidated. Consolidation means merging two or more extents into one combined extent. For files that are entirely in direct extents, the extent reorganizer will attempt to consolidate extents into extents up to size **largesize**. All files of size **largesize** or less typically will be contiguous in one extent after reorganization. Since most files are small, this will usually include about 98 percent of all files.

The **Consolidatable Blocks** column contains the total number of blocks in **Consolidatable Extents**.

The **Immovable Extents** column contains the total number of extents that are considered to be immovable. In the report, an immovable extent appears in the allocation unit of the extent itself, as opposed to in the allocation unit of its inode. This is because the extent is considered to be immovable, and thus permanently fixed in the associated allocation unit.

The **Immovable Blocks** column contains the total number of blocks in immovable extents.

The figures under the Free Extents by Size heading indicate per-allocation unit totals for free extents of each size. The totals are for free extents of size 1, 2, 4, 8, 16, . . . up to a maximum of the number of data blocks in an allocation unit. The totals should match the output of **df -o s** unless there has been recent allocation or deallocation activity (as this utility acts on

mounted file systems). These figures give an indication of fragmentation and extent availability on a per-allocation-unit basis. For each allocation unit, and for the complete file system, the total free blocks and total free blocks by category are shown.

The figures are presented as follows:

- The **Free Blocks** figure indicates the total number of free blocks.
- The **Smaller Than 8** figure indicates the percentage of free blocks that are in extents less than 8 blocks in length.
- The **Smaller Than 64** figure indicates the percentage of free blocks that are in extents less than 64 blocks in length.

In the preceding example, 4 percent of free space is in extents less than 8 blocks in length, and 24 percent of the free space is in extents less than 64 blocks in length. This represents a typical value for a mature file system that is regularly reorganized. The total free space is about 10 percent.

Extent Reorganization

If the **-e** option is specified, **fsadm** will reorganize the data extents on the file system whose mount point is **mountpoint_dir**. The primary goal of extent reorganization is to defragment the file system.

To reduce fragmentation, extent reorganization tries to place all small files in one contiguous extent. The **-l** option is used to specify the size of a file that is considered large. The default is 64 blocks. Extent reorganization also tries to group large files into large extents of at least 64 blocks. In addition to reducing fragmentation, extent reorganization improves performance. Small files can be read or written in one I/O operation. Large files can approach raw-disk performance for sequential I/O operations.

Extent reorganization also tries to improve the locality of reference on the file system. Extents are moved into the same allocation unit as their inode. Within the allocation unit, small files and directories are migrated to the front of the allocation unit. Large files and inactive files are migrated towards the back of the allocation unit. (A file is considered inactive if the access time on the inode is more than 14 days old. The time interval can be varied using the **-a** option.) Extent reorganization should reduce the average seek time by placing inodes and frequently used data closer together.

fsadm will try to perform extent reorganization on all inodes on the file system. Each pass through the inodes will move the file system closer to the organization considered optimal by **fsadm**. The first pass might place a file into one contiguous extent. The second pass might move the file into the same allocation unit as its inode. Then, since the first file has been moved, a third pass might move extents for a file in another allocation unit into the space vacated by the first file during the second pass.

When the file system is more than 90 percent full, **fsadm** shifts to a different reorganization scheme. Instead of attempting to make files contiguous, extent reorganization tries to defragment the free-extent map into chunks of at least 64 blocks or the size specified by the **-l** option.

The following is the command line to perform extent reorganization:

```
fsadm -F vxfs -e [-sv] [-p passes] [-t time] [-a days] [-l largesize] /mountpoint_dir
```

The following example illustrates the output from the **fsadm -F vxfs -e -s** command:

```
# fsadm -F vxfs -e -s
```

Allocation Unit 0, Pass 1 Statistics

		Extents Searched	Consolidations Number	Extents	Performed Blocks	Total Errors File Busy	Not Free
au	0	2467	11	30	310	0	0
au	1	0	0	0	0	0	0
au	2	0	0	0	0	0	0
au	3	0	0	0	0	0	0
au	4	0	0	0	0	0	0
total		2467	11	30	310	0	0

		In Proper Location Extents	Blocks	Moved to Proper Location Extents	Blocks
au	0	1379	8484	794	10925
au	1	0	0	0	0
au	2	0	0	0	0
au	3	0	0	0	0
au	4	0	0	0	0
total		1379	8484	794	10925

		Moved to Free Area Extents	Blocks	In Free Area Extents	Blocks	Could not be Moved Extents	Blocks
au	0	231	4851	4	133	0	0
au	1	0	0	0	0	0	0
au	2	0	0	0	0	0	0
au	3	0	0	0	0	0	0
au	4	0	0	0	0	0	0
total		231	4851	4	133	0	0

Allocation Unit 0, Pass 2 Statistics

		Extents Searched	Consolidations Number	Extents	Performed Blocks	Total Errors File Busy	Not Free
au	0	2467	0	0	0	0	0
au	1	0	0	0	0	0	0
au	2	0	0	0	0	0	0
au	3	0	0	0	0	0	0
au	4	0	0	0	0	0	0
total		2467	0	0	0	0	0

		In Proper Location	Moved to Proper Location
--	--	--------------------	--------------------------

Module 10
VxFS Performance Issues

		Extents	Blocks	Extents	Blocks
au	0	2173	19409	235	4984
au	1	0	0	0	0
au	2	0	0	0	0
au	3	0	0	0	0
au	4	0	0	0	0
total		2173	19409	235	4984

		Moved to Free Area		In Free Area		Could not be Moved	
		Extents	Blocks	Extents	Blocks	Extents	Blocks
au	0	0	0	0	0	0	0
au	1	0	0	0	0	0	0
au	2	0	0	0	0	0	0
au	3	0	0	0	0	0	0
au	4	0	0	0	0	0	0
total		0	0	0	0	0	0

Note that the default five passes were scheduled, but the reorganization finished in two passes.

This file system had not had much activity since the last reorganization, with the result that little reorganization was required. The time it takes to complete extent reorganization varies, depending on fragmentation and disk speeds. However, in general, extent reorganization may be expected to take approximately one minute for every 10 megabytes of disk space used.

In the preceding example, the **Extents Searched** column contains the total number of extents examined. The **Number** column (located under the **Consolidations Performed** heading) contains the total number of consolidations or merging of extents performed. The **Extents** column (located under the **Consolidations Performed** heading) contains the total number of extents that were consolidated. (More than one extent may be consolidated in one operation.) The **Blocks** column (located under the **Consolidations Performed** heading) contains the total number of blocks that were consolidated.

The **File Busy** column (located under the **Total Errors** heading) contains the total number of reorganization requests that failed because the file was active during reorganization. The **Not Free** column (located under the **Total Errors** heading) contains the total number of reorganization requests that failed because an extent that the reorganizer expected to be free was allocated at some time during the reorganization.

The **In Proper Location** column contains the total extents and blocks that were already in the proper location at the start of the pass. The **Moved to Proper Location** column contains the total extents and blocks that were moved to the proper location during the pass.

The **Moved to Free Area** column contains the total number of extents and blocks that were moved into a convenient free area in order to free up space designated as the proper location for an extent in the allocation unit being reorganized. The **In Free Area** column contains the total number of extents and blocks that were in areas designated as free areas at the beginning of the pass.

The **Could not be Moved** column contains the total number of extents and blocks that were in an undesirable location and could not be moved. This occurs when there is not

enough free space to allow sufficient extent movement to take place. This often occurs on the first few passes for an allocation unit if a large amount of reorganization needs to be performed.

If the next to last pass of the reorganization run indicates extents that cannot be moved, then the reorganization fails. A failed reorganization may leave the file system badly fragmented, since free areas are used when trying to free up reserved locations. To lessen this fragmentation, extents are not moved into the free areas on the final two passes of the extent reorganizer and the last pass of the extent reorganizer only consolidates free space.

To defragment a BaseJFS you need to perform the same steps you would for an HFS:

1. Back up the file system (with **fbackup**).
2. Make a new file system (with **newfs**).
3. Restore the data from tape (with **frecover**).

10-18. SLIDE: Using `setext`

Using `setext`



The `setext` command can manipulate the extent allocation policies of the JFS file system on a file by file basis:

- Use `setext` to override default VxFS extent allocation policies
- Specify the extent size
- Force files to be continuous
- Pre-reserve space for future continuous growth
- Prevent files from growing past reservation
- Use `getext` to view file parameters
- Use `ls -le` to view extent parameters

Student Notes

`setext` specifies a fixed extent size for a file, and reserves space for a file. The file must already exist.

```
setext [-F vxfs] [-e extentsize] [-r reservation] [[-f flag]... ] file
```

Options:

<code>-e extentsize</code>	Specify fixed extent size (in file system blocks)
<code>-r reservation</code>	Pre-allocate space (in file system blocks)
<code>-f align</code>	All extents aligned on <i>extentsize</i> boundaries relative the start of allocation units
<code>-f contig</code>	Reservation must be allocated contiguously
<code>-f noextend</code>	File may not be extended once pre-allocated space has been used
<code>-f chgsize</code>	Reservation is incorporated into file; on-disk inode updated with size and block count information that includes the reserved space

-f noreserve	Reservation made as non-persistent allocation to file; on-disk inode not updated; associated with file until last close, then trimmed to current file size
-f trim	Reservation is trimmed to current file size upon last close by all processes that have the file open

Example using **setext**

```
# touch bigfile.0 bigfile.1 bigfile.2
# /usr/sbin/setext -F vxfs -r 4096 -f contig bigfile.1
# /usr/sbin/setext -F vxfs -f align -e 128 bigfile.2

# cp bigfile bigfile.0
# cp bigfile bigfile.1
# cp bigfile bigfile.2

# ls -l bigfile* # ls -l bigfile*
-rw-r--r-- 1 root other 2691000 Nov 2 10:52 bigfile.0
-rw-r--r-- 1 root other 2691000 Nov 2 10:53 bigfile.1
-rw-r--r-- 1 root other 2691000 Nov 2 10:53 bigfile.2

# /usr/sbin/getext -F vxfs bigfile.*
bigfile.0: Bsize 1024 Reserve 0 Extent Size 0
bigfile.1: Bsize 1024 Reserve 4096 Extent Size 0
bigfile.2: Bsize 1024 Reserve 0 Extent Size 128
```

Example output from **ls -le**

```
# ls -le bigfile*
-rw-r--r-- 1 root other 2691000 Nov 2 10:52 bigfile.0
-rw-r--r-- 1 root other 2691000 Nov 2 10:53 bigfile.1 :res 4096 ext 0
-rw-r--r-- 1 root other 2691000 Nov 2 10:53 bigfile.2 :res 0 ext 128
```

10-19. SLIDE: I/O Tunable Parameters

I/O Tunable Parameters



- JFS provides a set of eleven (11) tunable I/O parameters.
- If the default I/O parameters are not acceptable, then the `/etc/vx/tunefstab` file can be used.
- `mount_vxfs(1M)` invokes the `vxtunefs(1M)` command to process the contents of the `/etc/vx/tunefstab` file.
- Failure to set I/O parameters does not prevent mount from occurring

Student Notes

JFS Tunable Parameters

I/O Parameter	Description
read_pref_io	The preferred read request size. The file system uses this in conjunction with the read_nstream value to determine how much data to read-ahead. Default value is 64K.
write_pref_io	The preferred write request size. The file system uses this in conjunction with the write_nstream value to determine how to do flush-behind on writes. Default value is 64K.
read_nstream	The number of parallel read requests of size read_pref_io to have outstanding at one time. The file system uses the product of read_nstream multiplied by read_pref_io to determine its read-ahead size. Default value for read_nstream is 1.

write_nstream The number of parallel write requests of size **write_pref_io** to have outstanding at one time. The file system uses the product of **write_nstream** multiplied by **write_pref_io** to determine when to do flush-behind on writes. Default value for **write_nstream** is 1.

(Only the first four parameters are described here. Refer to man page for vxtunefs(1m) for the remainder.)

10-20. SLIDE: `vxtunefs` Command for Tuning VxFS

vxtunefs Command for Tuning VxFS



```
# vxtunefs /tondir
Filesystem i/o parameters for /tondir
read_pref_io = 65536 # Preferred read request size is 64k
read_nstream = 1 # Desired number of parallel read_pref_io's
read_unit_io = 65536
write_pref_io = 65536 # Preferred write request size 64k
write_nstream = 1 # Desired number of parallel write_pref_io's
write_unit_io = 65536
pref_strength = 10
buf_breakup_size = 131072
discovered_direct_iosz = 262144 # Large I/O's treated like direct for speed
max_direct_iosz = 131072
default_indir_size = 8192
gio_cache_enable = 0
max_diskq = 1048576
initial_extent_size = 8
max_seqio_extent_size = 2048
max_buf_data_size = 8192
```

Student Notes

The slide shows the output of the `vxtunefs` command being used to query the configuration of a VxFS file system.

`vxtunefs` Command Details

```
/sbin/vxtunefs [-ps] [-f tunefstab] [-o parameter=value]
[{{mount_point|block_special}}]...
```

Options:

- | | |
|---------------------------------|--|
| <code>-f filename</code> | Use filename instead of <code>/etc/vx/tunefstab</code> as the file containing tuning parameters. |
| <code>-o parameter=value</code> | Specify parameters for the file systems listed on the command line. The parameters are listed below. |
| <code>-p</code> | Print the tuning parameters for all the file systems specified on the command line. |

-s Set the new tuning parameters for the VxFS file systems specified on the command line or in the `tunefstab` file.

`vxtunefs` sets or prints tunable I/O parameters of mounted file systems. `vxtunefs` can set parameters describing the I/O properties of the underlying device, parameters to indicate when to treat an I/O as direct I/O, or parameters to control the extent allocation policy for the specified file system.

With no options specified, `vxtunefs` prints the existing VxFS parameters for the specified file systems.

`vxtunefs` works on a list of mount points specified on the command line, or all the mounted file systems listed in the `tunefstab` file. The default `tunefstab` file is `/etc/vx/tunefstab`. You can change the default using the `-f` option.

`vxtunefs` can be run at any time on a mounted file system, and all parameter changes take immediate effect. Parameters specified on the command line override parameters listed in the `tunefstab` file.

If `/etc/vx/tunefstab` exists, the VxFS-specific mount command invokes `vxtunefs` to set device parameters from `/etc/vx/tunefstab`.

10-21. SLIDE: /etc/vx/tunefstab Configuration

/etc/vx/tunefstab Configuration



- File is read every time a VxFS is mounted.
- Automatic permanent vxtunefs options implemented here
- File format as follows:


```
block-device    tunefs-options
system-default  tunefs-options
```
- Options set for individual file systems or globally all VxFS file systems

Student Notes

The **tunefstab** file contains tuning parameters for VxFS file systems. **vxtunefs** sets the tuning parameters for mounted file systems by processing command line options or by reading parameters in the **tunefstab** file.

Each entry in **tunefstab** is a line of fields in one of the following formats:

```
block-device    tunefs-options
system-default  tunefs-options
```

block-device is the name of the device on which the file system exists. If there is more than one line that specifies options for a device, each line is processed and the options are set in order.

In place of **block-device**, **system-default** specifies tunables for each device to process. If there is an entry for both a block device and a system default, the system default value takes precedence.

Lines in **tunefstab** that start with the pound (#) character are treated as comments and ignored.

The `tunefs`-options correspond to the tuneable parameters that `vxtunefs` and `mount_vxfs` set on the file system. Each option in this list is a name=value pair. Separate the options by commas, with no spaces or tabs between the options and commas.

See the `vxtunefs(1M)` manual page for a description of the supported options.

Examples

If you have a four column striped volume, `/dev/vg01/lvol3`, with a stripe unit size of 128 kilobytes per disk, set the `read_pref_io` and `read_nstream` parameters 128 and four, respectively. You can do this in two ways:

```
/dev/vg01/lvol3          read_pref_io=128k,read_nstream=4
```

or:

```
/dev/vg01/lvol3          read_pref_io=128k
/dev/vg01/lvol3          read_nstream=4
```

To set the discovered direct I/O size so that it is always lower than the default, add the following line to the `/etc/vx/tunefstab` file:

```
/dev/dsk/c3t1d0          discovered_direct_iosz=128K
```

10-22. SLIDE: Taking Snapshots and Performance

Taking Snapshots and Performance



- Issues for the “online” Snapped File System:
 - Read performance should not be affected.
 - Any writes after the snap will be 2-3 times slower
 - Subsequent writes to the same area will perform normally
 - Have the snapshot on a separate physical disk
 - Tests of OLTP show 15-20% degradation
- Issues for the “backup” Snapshot File System
 - Snapshot performance should be equivalent to normal JFS
 - Read performance suffers if Snapped (online) half is busy

Student Notes

Performance of the “Advanced” (Snapped) File System.

The write performance of the online (snapped) file system will be degraded but the read performance will stay the same. It is important to ensure that the snapshot file system (the backup) resides on a different physical disk, otherwise backup I/O will use up valuable bandwidth.

Initial writes to a block after the snapshot is started will be 2 to 3 times slower.

1. Read the old data
2. Write the old data to the snapshot
3. Write of the new data

Multiple snapshots would cause this process to be even slower.

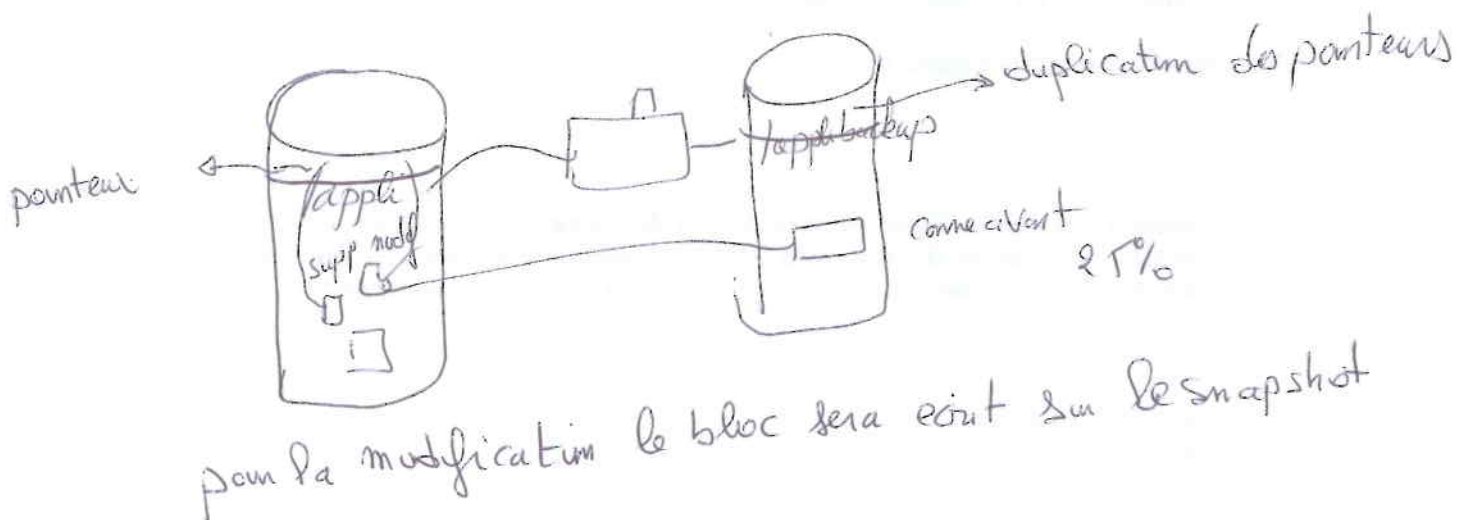
Only the initial write suffers, subsequent changes are not recorded in the snapshot and therefore would proceed at normal speed.

Overall impact will depend on the read to write ratio and the mixing of the I/O operations. For example, Oracle running an OLTP workload on a snapped file system was measured about 15 to 20% slower than a file system that was not being snapped.

Performance of the "Backup" (Snapshot) File System.

Performance of the snapshot is maximized at the expense of writes to the snapped file system.

Reads from a snapshot file system will typically be at the same rate as from a normal JFS file system, allowing backups to proceed at the full speed of JFS. Reads from the snapshot are impacted if the snapped file system is very busy. Remember the read data comes from the snapped file system unless it has been modified.



*regarder db_block_size pour savoir l'état de bloc k
de vos fichiers*

10-23. LAB: JFS File System Tuning

Directions

The following lab exercise compares performance of JFS with different mount options. The mount options used with JFS can have a big impact on JFS performance.

1. Mount a JFS file system to be used for this lab under `/vxfs`.

```
# mount /dev/vg00/vxfs /vxfs
```

2. Because the above `mount` command specified no special mount options, the default mount options are used. Use the `mount -v` command to view the default options, including the option for transaction logging type.

What type of transaction logging does JFS use by default?

3. Change directory to `/vxfs`. Time the execution of the `disk_long` program, which writes 400 MB of data to the file system in 20 MB increments. After each 20 MB is written, the files are deleted. Run the command three times and record the middle results.

```
# cd /vxfs
```

```
# timex ./disk_long
# timex ./disk_long
# timex ./disk_long
```

Record middle results:

Real: _____ User: _____ Sys: _____

4. Remount the JFS file system using `delaylog` option. This helps performance of noncritical transactions. Run the command three times and record the middle results.

```
# cd /
# umount /vxfs
# mount -o delaylog /dev/vg00/vxfs /vxfs
# cd /vxfs
# timex ./disk_long
# timex ./disk_long
# timex ./disk_long
```

Record middle results:

Real: _____ User: _____ Sys: _____

Based on the results, does the `disk_long` program perform many noncritical transactions?

5. Remount the JFS file system using `tmplog` option. This causes the system call to return *after* the JFS transaction is updated in memory (step 1 from lecture), and *before* the transaction is written to the intent log. Run the command three times and record the middle results.

```
# cd /
# umount /vxfs
# mount -o tmplog /dev/vg00/vxfs /vxfs
# cd /vxfs
# timex ./disk_long
# timex ./disk_long
# timex ./disk_long
```

Record middle results:

Real: _____ User: _____ Sys: _____

Based on the results, why does the `disk_long` program show little improvement when mounted with `tmplog`?

6. Remount the JFS file system using `tmpcache` option. This allows the JFS transaction to be created without having to wait for the user data to be written in extending write calls. Run the command three times and record the middle results.

```
# cd /
# umount /vxfs
# mount -o mincache=tmpcache /dev/vg00/vxfs /vxfs
# cd /vxfs
# timex ./disk_long
# timex ./disk_long
# timex ./disk_long
```

Record middle results:

Real: _____ User: _____ Sys: _____

Module 10
VxFS Performance Issues

7. Remount the JFS file system using **direct** option. This option requires all user data and all JFS transactions to bypass the buffer cache and go directly to disk. Run the command just once and record the results.

```
# cd /  
# umount /vxfs  
# mount -o mincache=direct /dev/vg00/vxfs /vxfs  
# cd /jfs  
# timex ./disk_long
```

Record results:

Real: _____ User: _____ Sys: _____

Based on the results, why does the **disk_long** program show poor performance results when mounted with **mincache=direct**? When would this option be appropriate to use?

8. Dismount the VxFS file system.

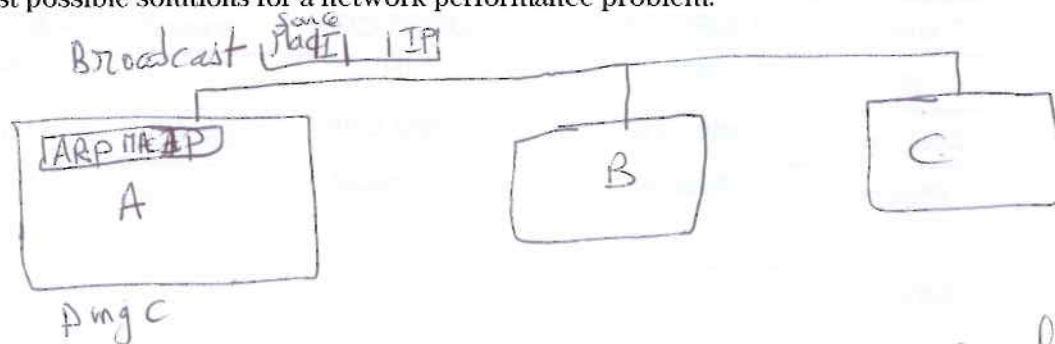
```
# umount /vxfs
```


Module 11 — Network Performance

Objectives

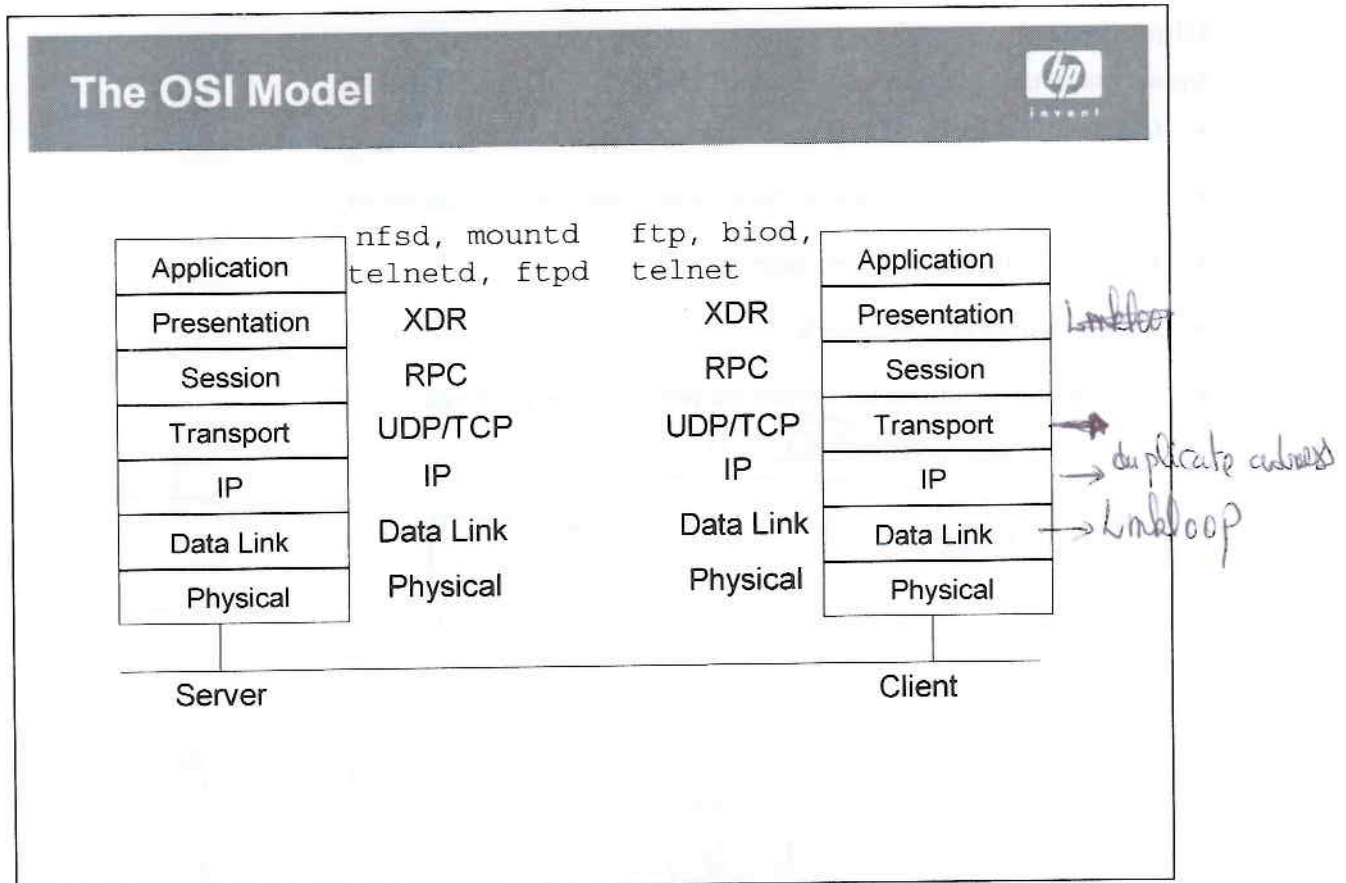
Upon completion of this module, you will be able to do the following:

- 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.



- 1) verification et envoi de IP pour récupérer la Mac de destination
- 2) envoi du ping sera envoyé en port (118 Exp)

11-1. SLIDE: The OSI Model



Student Notes

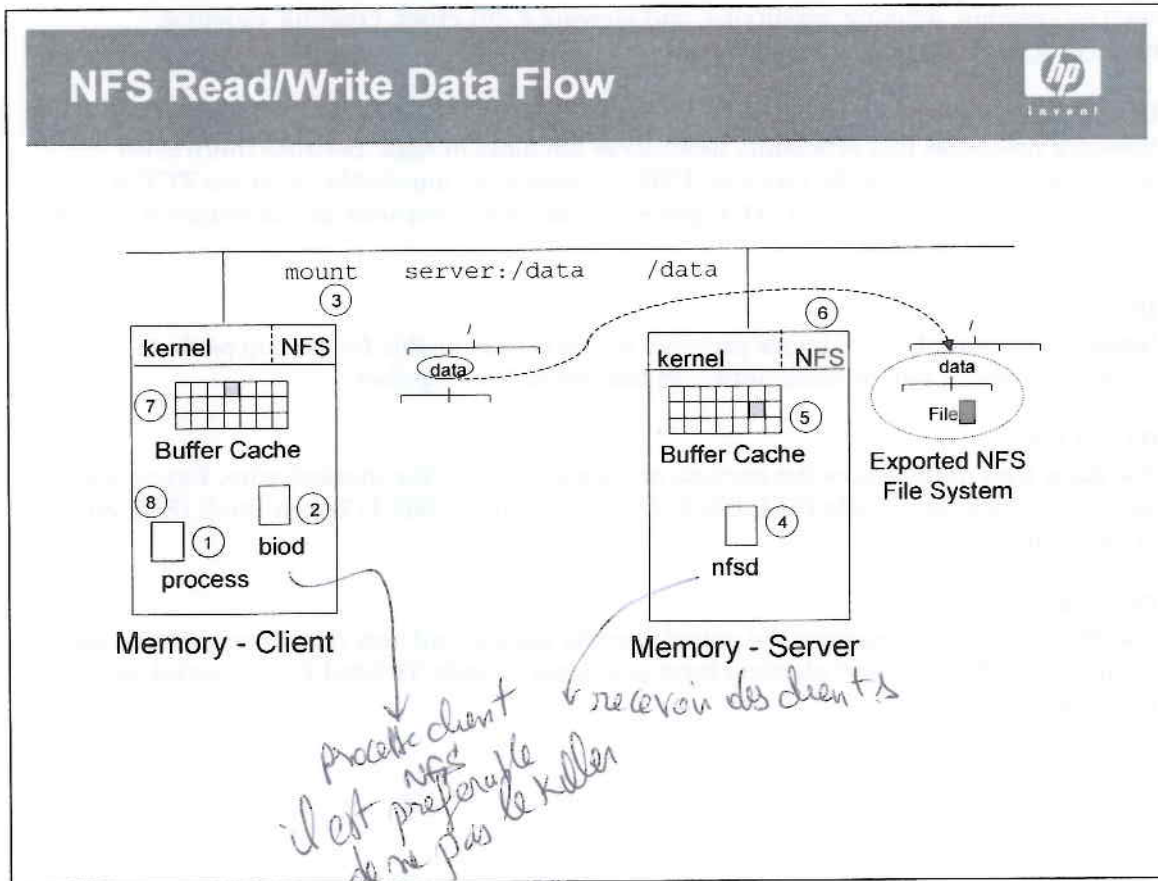
Networking allows one computer (server) to communicate with and share its local files and directories with other computers (clients) in a homogeneous environment.

Network Protocols

- **NFSD, MOUNTD, FTPD, TELNETD**
The **Networking server daemons** respond to requests from clients and perform the requested operations.
- **BIOD, FTP, TELNET**
The **Networking user applications** request operations to be performed for them on the server.
- **XDR**
External data representation is a machine-independent data format used by applications to translate machine-dependent data formats to a universal format that can be used by other networking hosts using XDR.

- **RPC/Session Layer**
The **remote procedure call** mechanism allows a server machine to define a procedure that a client program can call. This is how a client can perform file system operations, such as creating, deleting, modifying, and viewing a directory, creating, deleting, modifying, and copying a file, and so on.
- **UDP/TCP**
Network protocols that efficiently move large amounts of data. Because there is no acknowledgement from the receiver, **UDP** is considered unreliable, whereas **TCP** is considered reliable. However, **TCP** generally has more overhead and therefore does not perform as well as **UDP**.
- **IP**
Internet protocol is a network protocol which is responsible for getting packets between hosts on one or more networks that are linked together.
- **Data Link**
The **data link** defines how the packets are assembled on the physical wire. Examples of data link protocols include IEEE 802.3 (CSMA/CD), IEEE 802.4 (Token Bus), IEEE 802.5 (Token Ring).
- **Physical**
The **Physical** layer describes the actual transfer media, and how data is transferred on the network. Examples of physical layer protocols include Twisted Pair, Coaxial, and Fiber Optics.

11-2. SLIDE: NFS Read/Write Data Flow



Student Notes

As a prime example of how network performance can affect applications, let's look at how NFS works. The above slide shows a high level overview of the sequence of events which occur when an NFS client attempts to access data on an NFS server:

1. A user process issues the **read()** system call against an NFS mounted file system. The user process goes into a wait state, waiting for the system call to return.
2. Upon checking the buffer cache for the requested data (assume data is not in the buffer cache), the **biode** daemon immediately follows the original read with a read-ahead request. This is done by **biode** so subsequent I/O requests have a better chance of being satisfied through the buffer cache.
3. The NFS subsystem within the kernel on the client issues an RPC read request on behalf of the process (and a second on behalf of **biode**) to the NFS server.
4. The NFS server receives the request and schedules an **nfsd** process to handle it.

5. The `nfsd` daemon performs the file system read and the data is returned to the `nfsd` daemon through the servers' buffer cache.
6. The NFS subsystem within the kernel on the server schedules a reply to the client containing the requested data.
7. The data is returned to the client process through the buffer cache on the client. The data, plus the data read ahead by the `biobd` daemon, is stored in both the client's and server's buffer caches to allow future I/O requests to come from the buffer caches.
8. The read system call is returned (along with the data) to the client process.

As you can see, NFS initiates a fair amount of traffic over the network. Other services, such as `telnet` and `ftp`, have their own performance profiles. Some are interactive and response time is important. Others are task-oriented and rely mostly on throughput.

NFS v1 } portmap
NFS v2 }
NFS v3 } rpcbind

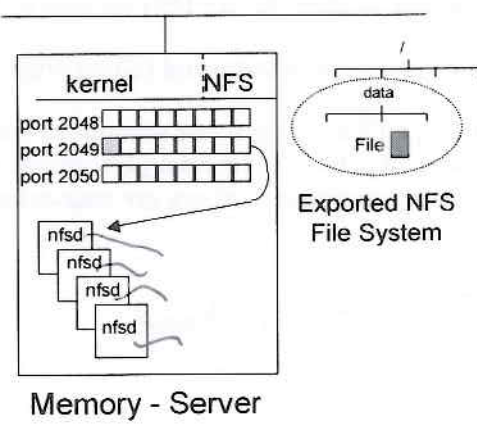
il faut que tous les clients aient la même version de
serveur pour éviter la lenteur ~~des fichiers~~ de réseau

11-3. SLIDE: NFS on HP-UX with UDP

NFS on HP-UX with UDP



- NFS packets arrive in the UDP socket (port 2049).
- The UDP socket is a 256-KB FIFO queue.
- The UDP socket is emptied by the `nfsd`s.
- Not enough `nfsd`s cause NFS packets to be backed up in the queue.



netstat -p udp -> overflow -> NFSD
pour savoir combien de NFS tant rpcinfo

Student Notes

NFS packets come into the NFS server through the UDP receive queue (port 2049). The size of this queue is 256 KB.

The NFS packets are processed sequentially, FIFO. Upon receipt of an NFS packet, an `nfsd` daemon is awakened, removes the request from the queue, and processes the request.

If requests come into the server faster than the daemons can process them, the UDP queue quickly begins to back up with requests. If the UDP queue is full when a new request arrives, the new request is dropped off the back of the queue. This is known as a **UDP socket overflow**. To prevent this, always have a sufficient number of daemons running.

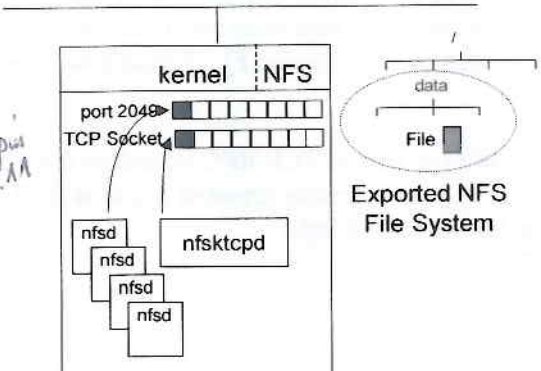
Regardless of how many `nfsd` daemons are running, only one will be awakened for each incoming request. This allows a site to meet the demands of peak workload without suffering performance problems during periods of light demand. NFS tuning can thus focus on file system and network performance than CPU performance, since the number of `nfsd` daemons does not impact performance.

11-4. SLIDE: NFS on HP-UX with TCP

NFS on HP-UX with TCP



- 16 `nfsd` processes are started by default.
- Multiple `nfsds` respond to udp queue
- Single multi thread `nfsktcpd` processes is dedicated to tcp
- Client establishes udp/tcp method on mounting.



Memory - Server

nfs → tcp -

nfs → udp

cad : nouveau car s'il trouve qu'il y'a le réseau lent.

l'envoi en tcp c'est qu'il est utilisé en local

le plus grand inconvénient udp

Student Notes

Network File System (NFS) is now supported over the connection-oriented protocol, TCP/IP for NFS versions 2 and 3, in addition to running over User Datagram Protocol (UDP). TCP transport increases dependability on wide-area networks (WANs). Generally, packets are successfully delivered more consistently because TCP provides congestion control and error recovery.

As a result, with this new functionality, NFS is now supported over WANs. As long as TCP is supported on the WAN, then NFS is supported also.

The `mount_nfs` command now supports a `proto=` option on the command line where the value for `proto` can be either UDP or TCP. (In the past, this option was ignored.) This change allows the administrator to specify which transport protocol they wish to use when mounting a remote file system.

If the `proto=` option is not specified, by default, NFS will attempt a TCP connection. If that fails, it will then try a UDP connection. Thus, by default, you will begin using TCP instead of UDP for NFS traffic when you begin using the 11i version of HP-UX. This should have little impact on you. You do, however, have the option to specify either UDP or TCP connections.

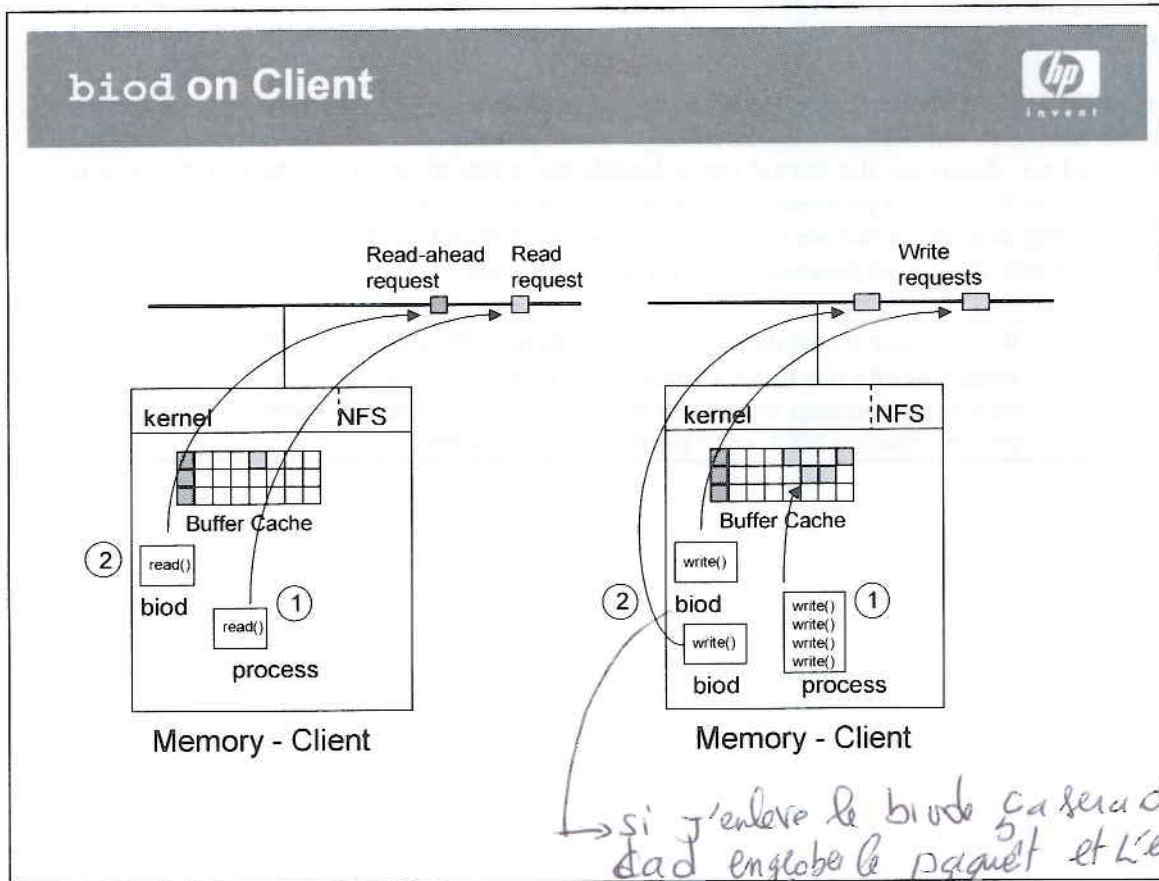
If you specify a `proto=` option, only the specified protocol will be attempted. If the server does not support the specified protocol, the mount will fail.

`nfsd` now opens TCP transport endpoints to receive incoming TCP requests. For TCP, the `nfsktcpd` is multi-threaded. For UDP, the `nfsd` is still multi-processed.

Kernel TCP threads execute under the process `nfsktcpd`. When counting the number of `nfsd` processes, keep in mind the following algorithm: An equal number of `nfsd`'s that support UDP will be created per processor and only one `nfsd` that supports TCP will be created. In the case of a four-way machine and `NUM_NFSDS=4` (set in `/etc/rc.config.d/nfsconf`), 17 `nfsd`'s will be created: 16 for UDP (4 per processor) and 1 for TCP.

`nfsstat` will now report TCP RPC statistics for both client and server. The TCP statistics will be under the connection-oriented tag and the UDP statistics will be under the connectionless-oriented tag.

11-5. SLIDE: biod on Client



Student Notes

The `biod` daemons allow performance on the NFS client to maintain the illusion of having file systems on the local disks. The `biod` daemons assist in improving NFS client performance by performing **read-aheads** and **write-behinds** for the client processes.

Read-Ahead Requests

The `biod` daemons help read performance on NFS clients by reading ahead (that is, prefetching) data into the buffer cache so that when the client needs the data, it will be in its buffer cache.

When an NFS client initiates a read request, and the data is not in its local buffer cache, the process performs the RPC read, itself. To prefetch data for the buffer cache, the kernel has the `biod` daemons send additional RPC read requests to the NFS server, just as if the NFS client process had requested this data. Subsequent read requests by the client (especially if reading sequentially) will find the data already in the buffer cache.

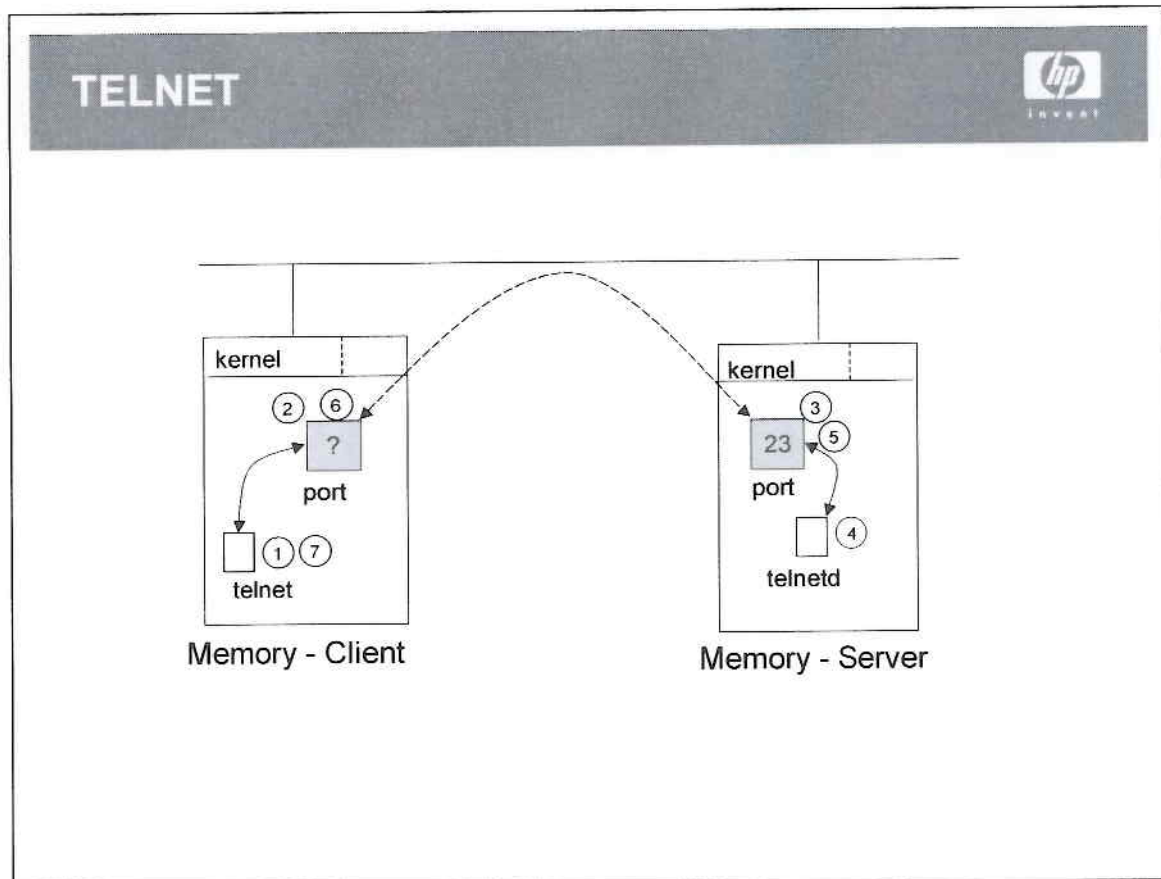
Write-Behind Requests

The **biobd** daemons assist in write performance by allowing the NFS client process performing the **write()** call to return immediately rather than waiting for the **write()** call to complete.

When an NFS client performs a **write()** call, the data is written to the client's buffer cache. Once the data is in the buffer cache, the kernel schedules an RPC write to occur. If there are available **biobd** daemons, the kernel can schedule the write to occur for the **biobd** daemons, rather than the NFS client process. This allows the client process to continue its execution without having to wait for the **write()** call to return. Instead of the client process waiting for the write call, the **biobd** daemon waits for the write call.

NOTE:	Without any biobd daemons on the client, NFS still works. The difference is no read-aheads are done, causing NFS read performance to suffer. All NFS clients performing writes are forced to wait for the RPC write requests to return, causing NFS write performance to suffer.
-------	--

11-6. SLIDE: TELNET



Student Notes

Telnet also uses sockets. A socket is simply a system-port pair. A connection is a pair of sockets.

On the client (when the user enters the `telnet` command), a port is assigned to the process from a pool of available ports. Thus a socket is formed on the client. A connection is established between that port and port 23 on the server (used exclusively to handle incoming `telnet` requests).

On the server (as a result of the connection), a `telnetd` daemon is spawned and linked to port 23.

Now, the `telnet` process running on the client (1) issues a request to execute some command on the server. The command is placed in a packet and sent through the socket on the client (2) to the socket on the server (3). The command is removed from the packet and given to the `telnetd` daemon (4) to execute.

The `telnetd` daemon executes the command and places the result in a packet. That packet is sent through the socket on the server (5) to the socket on the client (6). The results are removed from the packet and sent to the `telnet` process (7).

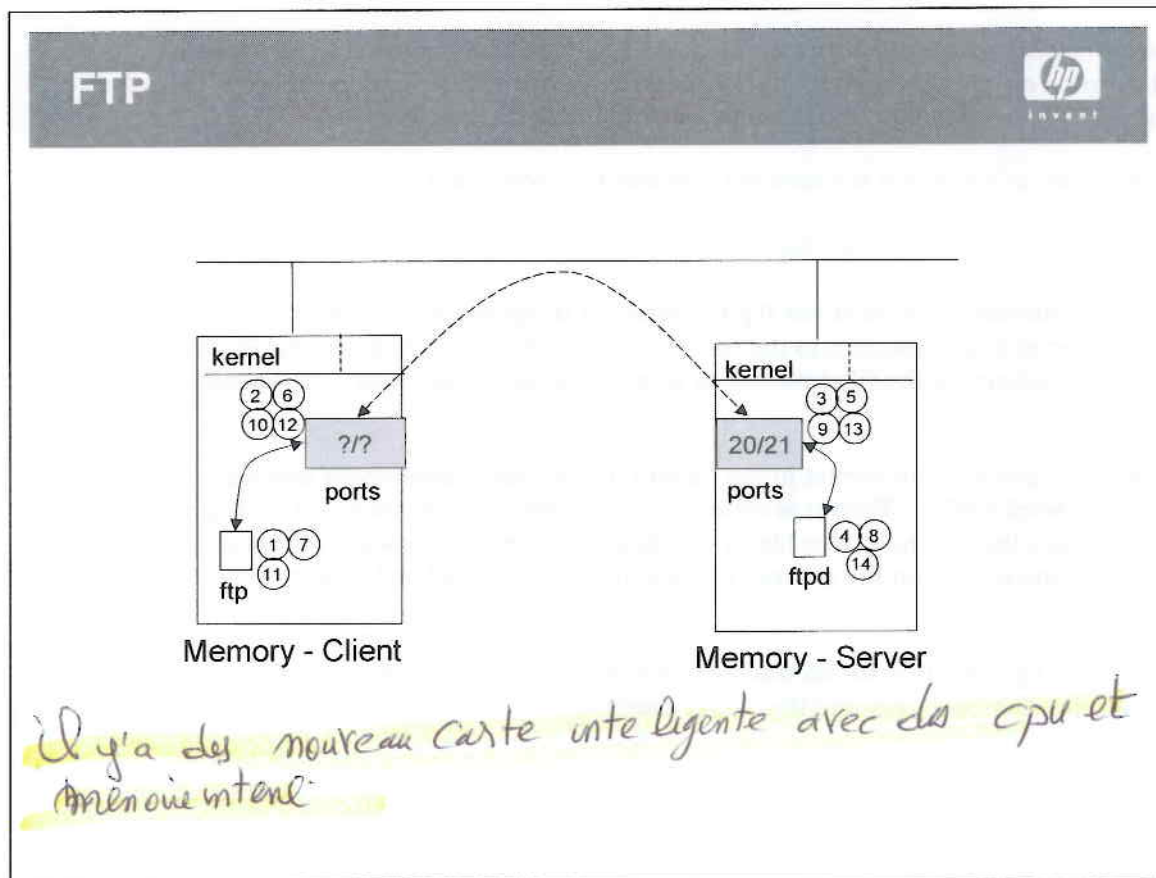
Module 11

Network Performance

By default, telnet uses TCP for its transfers, since it needs to establish a firm connection between the client process and the server daemon.



11-7. SLIDE: FTP



Student Notes

FTP also uses sockets. It uses a pair of connections to perform all its operations – one connection passes the commands and their results back and forth while the other connection passes file data back and forth.

On the client (when the user enters the `ftp` command), a port is assigned to the process from a pool of available ports. Thus a socket is formed on the client. A connection is established between that port and port 21 on the server (used exclusively to handle incoming `ftp` requests).

On the server (as a result of the connection), a `ftpd` daemon is spawned and linked to port 21.

Now, the `ftp` process running on the client (1) issues a request to execute some command on the server. The command is placed in a packet and sent through the socket on the client (2) to the socket on the server (3). The command is removed from the packet and given to the `ftpd` daemon (4) to execute.

Module 11

Network Performance

The **ftpd** daemon executes the command and places the result in a packet. That packet is sent through the socket on the server (5) to the socket on the client (6). The results are removed from the packet and sent to the **ftp** process (7).

If the command involves the transfer of some file data, the **ftp** process on the client (or the **ftpd** daemon on the server) initiates the transfer of the data from one socket to the other – using port 20 on the server and another available port on the client.

For example, let's say that the user entered the **ftp** command:

```
get /etc/hosts /tmp/hosts
```

When the command arrives at the **ftpd** daemon, it triggers a read of the **/etc/hosts** file from the server's file system into the server's buffer cache. Once there, the daemon (8) places the contents of the file into one or more packets (as necessary) and sends them to port 20 (9).

The packets arrive at the socket in the client (10) and are reassembled into the image of the file in a network buffer. Then it is copied into a buffer in buffer cache. The **ftp** process (11) acknowledges the receipt of the file by sending a packet to the socket (12) across the network to the socket on the server (13), where it is extracted and sent on to the daemon (14).

By default, **ftp** uses TCP for its transfers, since it needs to establish two firm connections between the client process and the server daemon.

11-8. SLIDE: Metrics to Monitor — NFS

Metrics to Monitor — NFS



- Number of `nfstd` daemons:
 - Monitor ratio between `calls`
 - Monitor CPU time used by all `nfstd` daemons
- Number of `biod` daemons:
 - Monitor number of waits due to no `biods` available
 - Monitor CPU time used by all `biod` daemons
- Number of `badcalls`
- Number of read and write NFS calls
- Number of UDP socket overflows, timeouts, retransmissions, and late responses

Student Notes

Number of `nfstd` daemons

Too few `nfstd` daemons can hinder performance on the NFS server. If all the `nfstd` daemons are busy when new NFS requests come in, then the requests have to wait until one of the daemons become free.

Monitor Ratio between `calls` and `nfstd` daemons

It is important to monitor the total RPC traffic (represented by the `RPC calls` field) relative to the NFS traffic (represented by the `nfstdrun` or `NFS calls` field). This can be especially helpful when there are multiple RPC-based applications (for example, NIS) running on the same system.

NOTE: The `nfstdrun` field is no longer present on the HP-UX 11.00 release, due to differences in how `nfstd` daemons are run in the 10.x release. To monitor the ratio of RPC to NFS traffic on HP-UX 11.00, use the `calls` fields.

Monitor CPU Time Used by All `nfsd` and `biod` Daemons

Distribution of NFS requests is spread over all `nfsd` daemons. This means each daemon is scheduled sequentially as NFS requests arrive. A sample CPU distribution on HP-UX would look something like:

CPU Utilization in minutes:

10				
9				
8				
7				
6				
5	X	X		
4	X	X	X	X
3	X	X	X	X
2	X	X	X	X
1	X	X	X	X
0	X	X	X	X
NFSDs	1	2	3	4

While the scheduling algorithm evenly balances the NFS call load across all `nfsd` daemons, it makes it difficult to determine if enough `nfsd` daemons are running on the server.

The number of `biod` daemons

Too many `biod` daemons can cause NFS servers to be flooded with requests, since each `biod` daemon can have an outstanding NFS request pending. Increasing the number of `biod` daemons on a client increases the number of NFS requests the client can have pending.

Too few `biod` daemons could mean an NFS request has to be performed by the client process itself (which means it has to wait) because no `biod` daemons are available. When the client process performs an RPC, the `wait` field (from the `nfsstat -c` command) is incremented by one. Too many waits indicate not enough `biod` daemons.

Number of Read/Write NFS Calls

The NFS read and NFS write RPC calls are the most resource-intensive of the NFS RPCs. Monitoring the percentage and quantities of these calls helps to give an indication of the total load these calls are placing on the NFS server.

Number of `nullrecv`

If the `nfsd` daemons are not being kept busy this counter will be incremented. If this counter is incrementing try reducing the number of `nfsd` daemons on the system until `nullrecv` is static.

Use `netstat -p udp` to View the Number of UDP Socket Overflows

UDP socket overflows can occur when too many NFS clients are sending requests to the NFS server, and too few `nfsd` daemons are running to handle the requests. When all the `nfsd` daemons are servicing RPC requests, none of them can read a new request from the UDP socket. Incoming RPCs are queued until the UDP socket structure becomes full. If the socket queue is full when a new request arrives, a UDP socket overflow condition occurs.

Number of badcalls

Bad calls indicate that the NFS server cannot process RPC requests. This could be due to authentication problems caused by having a user in too many groups, attempts to access exported file systems as root, or an improper secure RPC configuration. This can also be due to the server being down, or soft-mounted NFS file systems timing out.

Number of Time-Outs, Retransmissions, and Late Responses

A time-out indicates that the RPC call did not complete within the expected time period. The late responses (also known as `badxid`) refer to the NFS server responding to the client *after* the time period has expired. If time-outs and late responses are approximately equal, it indicates a healthy network, but an overloaded NFS server. If time-outs are high and late responses are low (or zero), it indicates packets are *never* making it to the server, and the network components (interface cards, cables, hubs) need to be examined.

11-9. SLIDE: Metrics to Monitor — Network

Metrics to Monitor — Network



- Amount of NFS traffic:
 - Monitor number of collisions
 - Monitor server workload
 - Monitor client workload
- Type of network topology and hardware
- Number of subnets
- Number of routers

Student Notes

Some key metrics to monitor from an overall network perspective include:

- **Amount of Traffic.** The amount of network traffic should be monitored across the entire LAN. However, unless network probes are available, this is very difficult to do. At a minimum, the amount of traffic into and out of the servers should be monitored with the `netstat` command.

When monitoring network traffic, it is important to know the maximum packets per second on the LAN. In the case of a 10 MB Ethernet, this would be:

$10 \text{ MB} / 8 \text{ bits_per_byte} = 1.2 \text{ MB per second}$

$1.2 \text{ MB} / 1 \text{ KB_average_packet_size} = 1,200 \text{ packets}$

$1,200 * 30\%_saturation_point = 360 \text{ packets}$

(Total MB per second)

(Total packet per second)

(Max packets per second with minimal collisions)

- **Type of Network Topology:** Each network topology has different limitations. Ethernet is the most common, but it is the slowest. More recent Ethernet technologies are faster, offering 100 Mbits/sec or even 1000 Mbits/sec. FDDI is the fastest, but it is somewhat expensive. Token Ring has no collision issues (since it is token based), but it is not as pervasive.
- **Number of Subnets:** Subnetting is a method for localizing traffic to help reduce packet congestion. If too much traffic exists on a network, it may need to be split into multiple subnets.
- **Number of Routers:** Routers are another possible solution to help segment network traffic. In addition, routers can help with network security issues and routing of diverse packet types.

11-10. SLIDE: Determining the NFS Workload

Determining the NFS Workload



- Run `nfsstat -s` or `netstat -c` to view total RPC calls
- Each week, zero counters; at end of week, divide total RPC calls by 5 days, 8 hours per day, 60 mins per hour, 60 seconds per minute
- Write a script to automate the data collection, calculation, and notification of the system administrator
- Set up a cron job to execute the script at the necessary times

Student Notes

The NFS workload on a server is defined as *the total number of NFS packets received and processed*.

The NFS workload on a client is defined as *the total number of NFS requests initiated from the client*.

It is important to establish a baseline regarding the NFS workload being placed on an NFS machine. This allows the system administrator to determine periods when the NFS workload is particularly high or low.

Sample Procedure for Calculating the NFS Workload

1. On Monday morning at 8:00 AM, run `nfsstat -z`. This zeroes out all the NFS registers.
2. On Friday evening at 5:00 PM, run `nfsstat -rs` (on the server) or `nfsstat -rc` (on the client). This shows the total number of NFS calls. Sample outputs are:

```
# nfsstat -rs
Server rpc:
calls      badcalls    nullrecv    badlen      xdrcall      nfsdrun
171792344  0                0           0           0           549734423

# nfsstat -rc

Client rpc:
  Connection oriented:
N/A
  Connectionless oriented:
calls      badcalls    retrans     badxid      timeouts    waits      newcreds
17547240    0           0           240         360         0          0
badverfs    timers      toobig      nomem       cantsend    buflocks
0           7           0           0           0           0
```

3. Calculate the average number of NFS calls per second by dividing the total RPC calls by 5 days, 8 hours per day, 60 minutes per hour, and 60 seconds per minute:

$(((((171792344 \text{ calls} / 5 \text{ days}) / 8 \text{ hours}) / 60 \text{ min}) / 60 \text{ sec}) = 1193 \text{ RPC calls/sec}$

Sample Script for Calculating an NFS Workload

A simple script for gathering and calculating the average number of NFS calls/second at the end of a week is shown below. Such a script could be called:

usr/local/bin/create_nfs_report

```
#!/usr/bin/sh
/usr/bin/nfsstat -rs | tail -1 | read calls trash
NFS_CALLS_PER_SEC=$(echo "$calls / 5 / 8 / 60 / 60" | bc)
HOST=$(hostname)
echo "The average NFS server calls (inbound) received for $(date +%x) was $NFS_CALLS_PER_SECOND on $HOST" | mailx -s "NFS Report" root

#!/usr/bin/sh
/usr/bin/nfsstat -rc | tail -3 | read calls trash
NFS_CALLS_PER_SEC=$(echo "$calls / 5 / 8 / 60 / 60" | bc)
HOST=$(hostname)
echo "The average NFS server calls (outbound) initiated for $(date +%x) was $NFS_CALLS_PER_SECOND on $HOST" | mailx -s "NFS Report" root
```

This report can be mailed to the performance analysis station or to the NFS machine.

Sample cron Entry for Automating the Procedure

To automate the process so that this happens every week without user invention, the following two entries can be placed in root's `crontab` file:

```
0 8 * * 1 /usr/sbin/nfsstat -z
0 17 * * 5 /usr/local/bin/create_nfs_report
```

This is a very simplistic form of data collection. Much more involved scripts can be developed. For example, scripts that take into account the time of day that demand is most heavy so that peak demand and demand patterns can be observed. Doing this on all NFS clients of an NFS server is also key.

It is important to establish a baseline regarding the NFS workload being placed initiated from all NFS clients. This allows the system administrator to determine periods when the NFS workload is particularly high or low.

11-11. SLIDE: NFS Monitoring — nfsstat Output

NFS Monitoring — nfsstat Output



```

# nfsstat -s

Connection oriented:
calls      badcalls    nullrecv    badlen      xdrcall     dupchecks   dupreqs
0          0           0           0           0           0           0

Connectionless oriented:
calls      badcalls    nullrecv    badlen      xdrcall     dupchecks   dupreqs
428        0           6           0           0           0           0

# nfsstat -c

Client rpc:
Connection oriented:
calls      badcall     badxids     timeouts    newcreds    badverfs    timers
0          0           0           0           0           0           0

cantconn   nomem       interrupts
0          0           0

Connectionless oriented:
calls      badcalls    retrans     badxids     timeouts    waits        newcreds
25345     304         1109        49          1410        0            0
badverfs   timers      toobig      nomem       cantsend    buflocks
0          16          0           0           0           0
  
```

Student Notes

The **nfsstat -s** report shows NFS statistics on an NFS server. The report shows overall RPC statistics and detailed NFS type packets received.

Fields of Interest in This Report

- calls (RPC)** This is the total RPC calls received. This should be compared to the total NFS calls received. Analyze the ratio of RPC calls to NFS calls to determine the percentage of RPC calls that are NFS related.
- nullrecv** This is the number of times an NFS daemon (or other RPC daemon) was scheduled to run only to find nothing in the UDP queue. This was very common on a 9.x system, since every time an NFS packet was placed in the UDP queue, all the **nfsd** daemons were awakened. The first **nfsd** daemon would take the NFS packet and the other daemons would find no packets in the UDP queue (incrementing the **nullrecv** field).

The example on the slide shows all the RPC packets received are NFS related. The six **nullrecvs** explain the difference between the RPC calls and NFS calls.

Module 11

Network Performance

The reason for the `nullrecv` may be due to a client retransmission duplicate request. For example, if a client sends an NFS read request and does not receive a response within its time-out period, it will re-send the same request, which causes a duplicate entry to be in the server's UDP queue. When the first `nfsd` daemon removes the first NFS read request, it will also remove the duplicate request. This causes the second `nfsd` daemon to find an empty UDP queue when it executes.

`nfsstat -s` (Full Output)

```
# nfsstat -s
```

```
Server rpc:
```

```
Connection oriented:
```

calls	badcalls	nullrecv
0	0	0
badlen	xdrCALL	dupchecks
0	0	0
dupreqs		
0		

```
Connectionless oriented:
```

calls	badcalls	nullrecv
0	0	0
badlen	xdrCALL	dupchecks
0	0	0
dupreqs		
0		

```
Server nfs:
```

calls	badcalls
0	0

```
Version 2: (0 calls)
```

null	getattr	setattr
0 0%	0 0%	0 0%
root	lookup	readlink
0 0%	0 0%	0 0%
read	wrcache	write
0 0%	0 0%	0 0%
create	remove	rename
0 0%	0 0%	0 0%
link	symlink	mkdir
0 0%	0 0%	0 0%
rmdir	readdir	statfs
0 0%	0 0%	0 0%

```
Version 3: (0 calls)
```

null	getattr	setattr
0 0%	0 0%	0 0%
lookup	access	readlink
0 0%	0 0%	0 0%
read	write	create
0 0%	0 0%	0 0%
mkdir	symlink	mknod
0 0%	0 0%	0 0%
remove	rmdir	rename
0 0%	0 0%	0 0%
link	readdir	readdir+


```

0 0%          0 0%          0 0%
fsstat        fsinfo        pathconf
0 0%          0 0%          0 0%
commit
0 0%

```

The **nfsstat -c** report shows NFS statistics on an NFS client. The report shows the amount of RPC calls generated by the client, as well as the specific NFS calls.

Fields of Interest in This Report

calls (RPC)	This is the total RPC calls generated by the client. This should be monitored relative to the total NFS calls generated. Analyze the ratio of RPC calls to NFS calls generated to determine the percentage of RPC calls that are NFS related.
waits	This is the number of times an NFS client process is put into a wait state due to no biobd daemons being available. An example of this would be during an NFS write. Normally, an NFS write is performed by a biobd daemon, and the biobd daemon waits for an acknowledgment to be returned by the NFS server. When no biobd daemons are available, then the actual client process itself performs the NFS write and the client has to wait for the acknowledgment to be returned.
timeouts	This is the number of times an NFS request was sent to the NFS server and no response was returned within the timeout period. A timeout can occur for two reasons: the NFS server machine is too busy and cannot get back to the client within the timeout period, or the network is having problems (collisions, bad interface card, bad hub) and the NFS request is never making it to the NFS server.
badxids	This indicates a bad or duplicate transfer ID number was returned from the NFS server. When a client sends a request to the server, a corresponding transfer ID is sent with the requests. When the NFS server responds, it specifies which transfer ID request it is responding to. For example, if a client sends an NFS request (with a corresponding transfer ID) and does not hear back from the NFS server, it then transmits the request a second time. If the NFS server returns the first and second requests after the client had timed out the first time, the client will view the second response as a duplicate transfer ID.

NOTE: The ratio of **timeouts** to **badxids** is an excellent way to determine if timeouts are occurring due to a slow NFS server or due to a failed network component. If the **badxids** are approximately the same as **timeouts**, then the NFS server is slow and the timeout period should be increased. If there are a lot of timeouts with few to no **badxids**, then the NFS requests are not making it to the server and there is most likely a failed LAN component.

Module 11

Network Performance

retrans This indicates the number of NFS requests retransmitted due to timeouts. Keep in mind, not every timeout causes a retransmission, as most clients error out after two to three retries.

badcalls This indicates an NFS request has reached its retry count and has returned an error. This is most often due to the NFS client not being able to reach the NFS server (either because the NFS server is down, or the network link between the client and server is down).

nfsstat -c (Full output)

```
# nfsstat -c

Client rpc:
Connection oriented:
calls          badcalls      badxids
0              0              0
timeouts       newcreds      badverfs
0              0              0
timers         cantconn      nomem
0              0              0
interrupts
0
Connectionless oriented:
calls          badcalls      retrans
55             0              0
badxids        timeouts      waits
0              0              0
newcreds       badverfs      timers
0              0              16
toobig         nomem         cantsend
0              0              0
bufulocks
0

Client nfs:
calls          badcalls      clgets
55             0              55
cltoomany
0
Version 2: (55 calls)
null           getattr       setattr
0 0%           50 90%        0 0%
root           lookup        readlink
0 0%           3 5%          0 0%
read           wr-cache      write
0 0%           0 0%          0 0%
create         remove        rename
0 0%           0 0%          0 0%
link           symlink        mkdir
0 0%           0 0%          0 0%
rmdir          readdir       statfs
0 0%           1 1%          1 1%
Version 3: (0 calls)
null           getattr       setattr
0 0%           0 0%          0 0%
```

Module 11
Network Performance

lookup	access	readlink
0 0%	0 0%	0 0%
read	write	create
0 0%	0 0%	0 0%
mkdir	symlink	mknod
0 0%	0 0%	0 0%
remove	rmdir	rename
0 0%	0 0%	0 0%
link	readdir	readdir+
0 0%	0 0%	0 0%
fsstat	fsinfo	pathconf
0 0%	0 0%	0 0%
commit		
0 0%		

11-12. SLIDE: Network Monitoring — lanadmin Output

Network Monitoring — lanadmin Output



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 = 0x800097bfb43
Administration Status (value) = up(1)
Operation Status (value) = up(1)
Last Change = 4834
Inbound Octets = 426550151
Inbound Unicast Packets = 3380123
Inbound Non-Unicast Packets = 1992200
Inbound Discards = 0
Inbound Errors = 1277
Inbound Unknown Protocols = 53618
Outbound Octets = 1653363768
Outbound Unicast Packets = 2626023
Outbound Non-Unicast Packets = 1454
Outbound Discards = 1
Outbound Errors = 0
Outbound Queue Length = 0
Specific = 655367

Press <Return> to continue

Ethernet-like Statistics Group

Index = 4
Alignment Errors = 0
FCS Errors = 0
Single Collision Frames = 6221
Multiple Collision Frames = 10151
Deferred Transmissions = 116267
Late Collisions = 0
Excessive Collisions = 0
Internal MAC Transmit Errors = 0
Carrier Sense Errors = 0
Frames Too Long = 0
Internal MAC Receive Errors = 0

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

Student Notes

The **lanadmin** command displays general network packet transmission statistics for a single system.

Fields of Interests in this Report

Collisions Frames These fields indicate the number of collisions detected by the system. Collisions slow NFS performance, as the network has to subside before any packets can be sent following a collision.

Inbound/Outbound This is the total of all packet types being sent and received from the **Packets** system. Compare this to the total number of daemon-related packets transmitted/received to obtain a ratio of total network traffic relative to the specific traffic.

The primary metric for determining if you have a network bottleneck is the ratio of collisions to out-bound packets. In this example, you would take the total number of collisions (6221 + 10151 = 16371) and divide it by the total number of outbound packets (2626023 + 1454 = 2627477) to get the percentage of collisions per outbound packet (16371 / 2627477 = 0.6%). The commonly used threshold is 5%. Any network experiencing greater than a 5% collision

rate is said to have a bottleneck. This system is well below that threshold. Of course, this metric only works on networks that experience collisions. Standard Ethernet does. Token rings do not.

The procedure for producing this report is:

1. Execute the `lanadmin` command.
2. From the main menu, select `lan`.
3. From the `lan` menu, select `display`.

Following is a complete output from this tool:

```
# lanadmin
```

```
LOCAL AREA NETWORK ONLINE ADMINISTRATION, Version 1.0
Thu, Mar 25,2004 11:22:51
```

```
Copyright 1994 Hewlett Packard Company.
All rights are reserved.
```

Test Selection mode.

```
lan      = LAN Interface Administration
menu     = Display this menu
quit     = Terminate the Administration
terse    = Do not display command menu
verbose  = Display command menu
```

Enter command: `lan`

LAN Interface test mode. LAN Interface PPA Number = 0

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

Enter command: `display`

```
LAN INTERFACE STATUS DISPLAY
Thu, Mar 25,2004 11:23:02
```

```
PPA Number           = 0
Description           = lan0 HP PCI 10/100Base-TX Core [100BASE-
TX,FD,
AUTO,TT=1500]
Type (value)         = ethernet-csmacd(6)
MTU Size              = 1500
Speed                 = 1000000000
Station Address       = 0x306e48c545
```

Module 11

Network Performance

```
Administration Status (value) = up(1)
Operation Status (value)      = up(1)
Last Change                    = 780
Inbound Octets                 = 1144058672
Inbound Unicast Packets       = 3513729
Inbound Non-Unicast Packets   = 2575374
Inbound Discards               = 0
Inbound Errors                 = 0
Inbound Unknown Protocols     = 13895
Outbound Octets                = 784916247
Outbound Unicast Packets      = 3600289
Outbound Non-Unicast Packets  = 379474
Outbound Discards              = 0
Outbound Errors                = 0
Outbound Queue Length         = 0
Specific                       = 655367
```

Press <Return> to continue <CR>

Ethernet-like Statistics Group

```
Index = 1
Alignment Errors = 0
FCS Errors = 0
Single Collision Frames = 0
Multiple Collision Frames = 0
Deferred Transmissions = 0
Late Collisions = 0
Excessive Collisions = 0
Internal MAC Transmit Errors = 0
Carrier Sense Errors = 0
Frames Too Long = 0
Internal MAC Receive Errors = 0
```

LAN Interface test mode. LAN Interface PPA Number = 0

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

Enter command: quit
#

11-13. SLIDE: Network Monitoring — netstat -i Output

Network Monitoring — netstat -i Output


```

# netstat -i
Name      Mtu  Network      Address      Ipkts  Ierrs  Opkts  Oerrs  Coll
lan0      1500  156.153.208.0  r265c75.cup.edunet.hp.com  4546682  0      4138618  0      0
lo0       4136  loopback      localhost    1178171  0      1178171  0      0

# netstat -p udp
udp:
  0 incomplete headers
  0 bad data length fields (Deleted from later versions)
  0 bad checksums
  0 socket overflows
  0 data discards (Deleted from later versions)

```

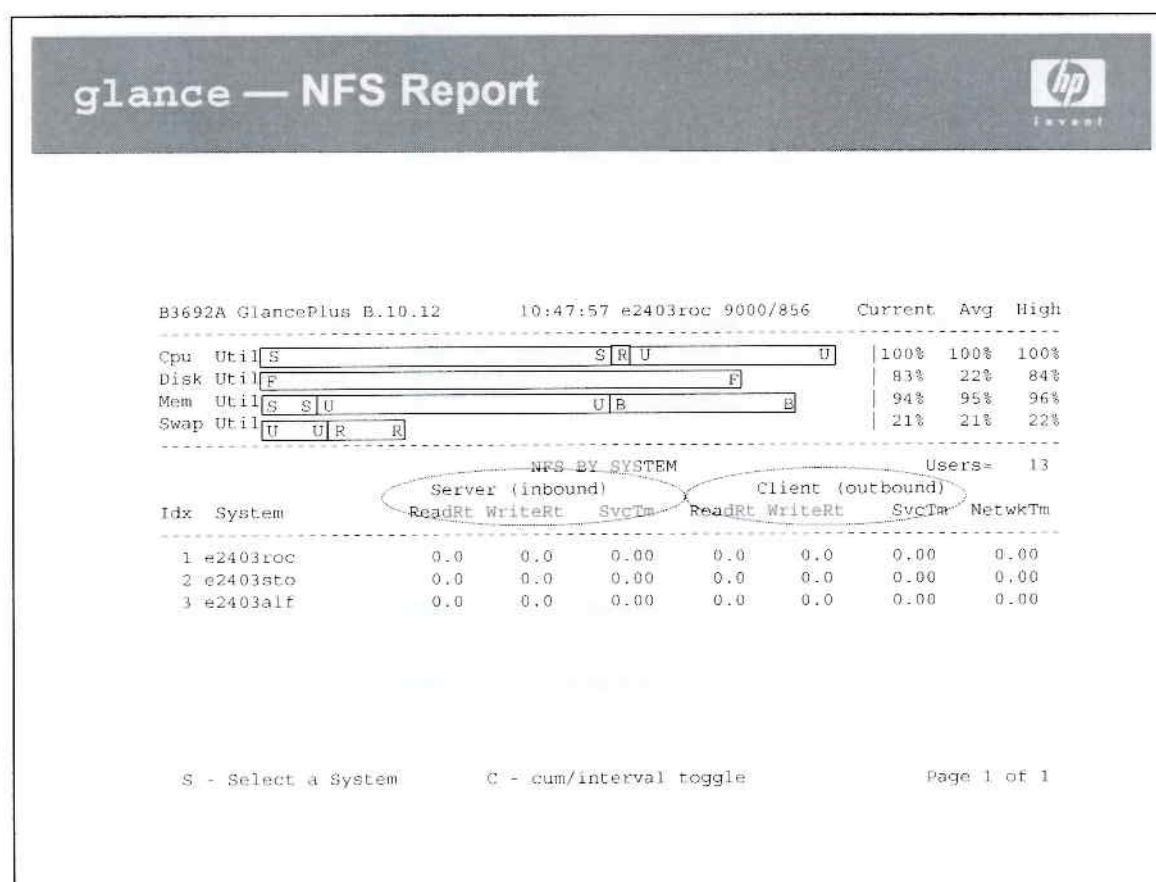
Student Notes

The **netstat** command can be used to monitor total collisions and total packet traffic in and out of a LAN card, as well as any UDP socket overflows.

The **-i** option monitors input packets, input errors, output packets, output errors, and collisions for every LAN card on the system. Some versions of this tool did not show the Input Errors, Output Errors, and the Collisions. The output of this tool can also be used to calculate the collision/outbound packet ratio described in the previous topic.

The **-p udp** option monitors overflows related to the UDP socket queue. If there are not enough **nfsd** daemons, the volume of incoming client NFS requests can exceed the server's ability to drain these requests from the UDP socket queue. When the socket queue becomes full and new NFS requests are received, the NFS request falls off the queue and a UDP socket overflow occurs.

11-14. SLIDE: glance — NFS Report



Student Notes

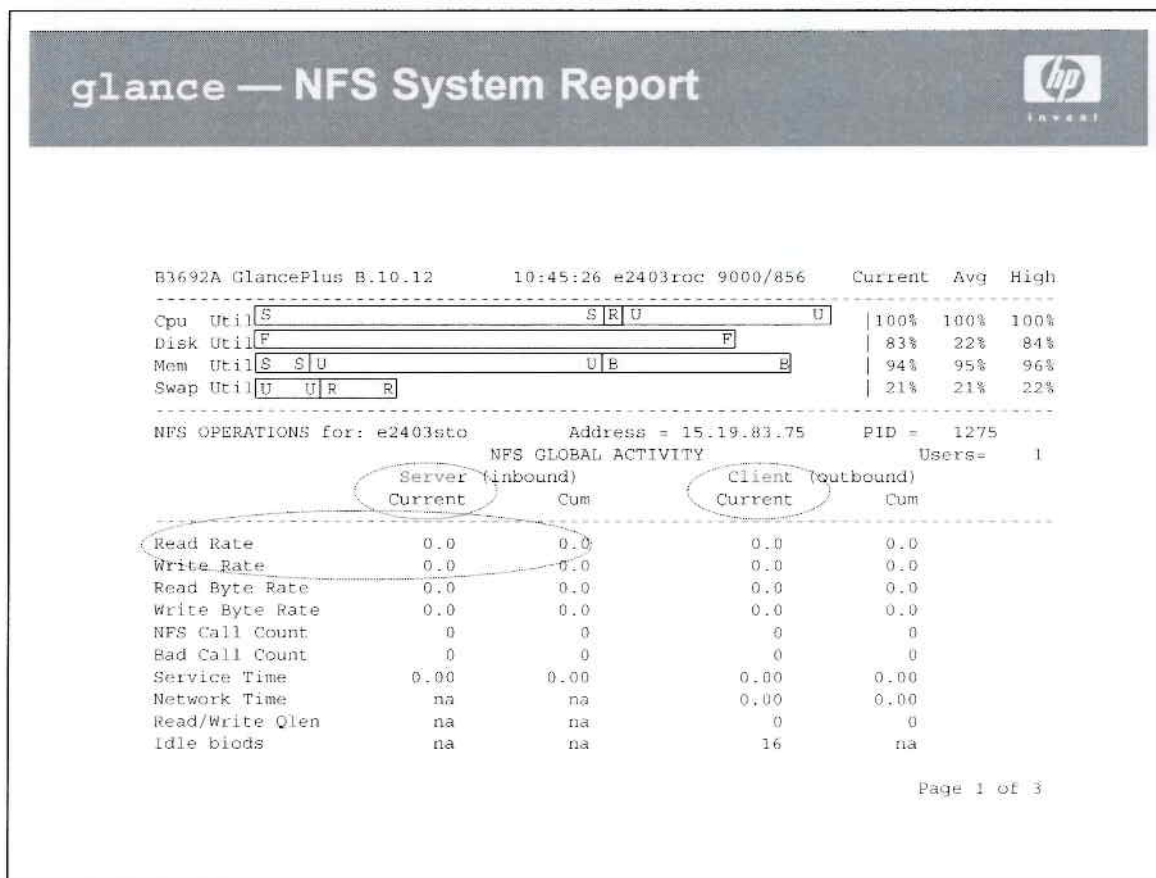
The **glance** NFS report (the **n** key) monitors total inbound requests for NFS servers and total outbound requests for NFS clients.

For NFS servers, the total number of inbound read/write requests received from each client is shown, along with the average amount of time for the server to service each request.

For NFS client systems, the total number outbound read/write requests sent to each NFS server is shown, along with the average amount of time, from the client perspective, for the requests to be serviced.

For a detailed inspection on the types of NFS requests being sent (client) or the types of NFS requests being received (server), the specific client or server can be selected with the **S** key.

11-15. SLIDE: glance — NFS System Report



Student Notes

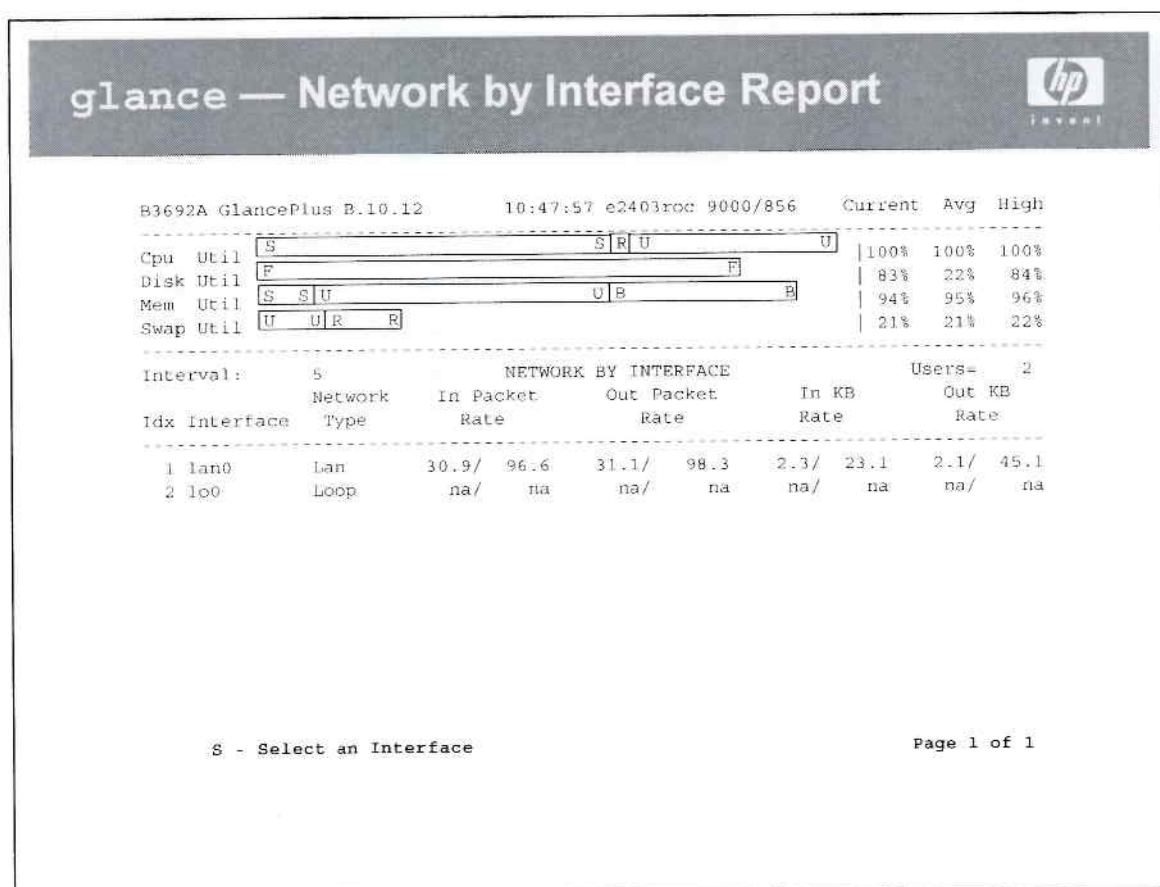
The **glance** NFS system report (the **N** key) displays the activity of NFS packets that are being received by an NFS server, or being sent as an NFS client.

If a system is both a client and a server, separate columns are maintained for each.

Fields of most interest in this report are the read and write rates, as these typically put the greatest load on a system.

Note that this is page one of three. On the following two pages, the individual **RPCs** are broken down by type and counted. There are version2 and version 3 counts to accommodate earlier and later versions of NFS.

11-16. SLIDE: glance — Network by Interface Report



Student Notes

The **glance** Network by Interface report (the 1 key) displays the activity of inbound and outbound packets.

Fields of most interest in this report are the inbound and outbound packets rates, as well as the KB transferred in and out by each network card. The lo0 interface is the internal loopback interface – used for diagnostics.

11-17. SLIDE: Tuning NFS

Tuning NFS



- Tune number of `nfsd` daemons
- Turn on "sticky bit" for exported executables
- Export file systems with asynchronous write option
- Avoid using symbolic links on exported file systems
- Tune number of `biod` daemons
- Tune mount options when mounting NFS file system:
 - `rsiz` and `wsiz` options
 - `retry` and `timeout` options

Asi j'ai un executable sur NFS server et si jamais un programme à exécuté alors il sera exécuté sur le client et ça si je mis le sticky bit

Student Notes

There are a number of NFS tuning solutions that can help to improve performance on NFS servers:

- **Tune number of `nfsd` daemons:** The default number of `nfsd` daemons in HP-UX 11.00 and earlier was four. This most likely is too small. The best recommendation for performance is to have two `nfsd` daemons for each simultaneous disk operation that can be performed. This allows a request to be received, while another is awaiting disk service. For example, on a system with four SCSI controllers and NFS-exported file systems spanning disks on these controllers, schedule eight `nfsd` daemons. In 11i, the default number of `nfsd` was raised to 16. This seems a more reasonable number.

The best indicator of too few `nfsd` daemons is UDP socket overflows. Increase the number of `nfsd` daemons if even one UDP socket overflow occurs. The size of the UDP socket queue can be viewed with the `netstat -an | grep udp | grep 2049` command.

Another indicator of too few `nfsd` server daemons is a high total of `badxids` being returned to NFS clients. Remember, only UDP requires the number of `nfsds` to be tuned. TCP uses multiple threads in the same daemon.

- **Turn on “sticky bit” for exported executables:** By default, text segments are not paged to swap, as their pages already exist on the file system. In the case of an executable program being loaded from an NFS server across the network to an NFS client, it is desirable to page the text locally, rather than return to the NFS server when the text page is needed again. This behavior can be achieved by setting the sticky bit to **ON** for the executable program.

Below is an example of setting the sticky bit to **ON** for an executable:

```
# chmod 1555 prgm
# ls -l prgm
-r-xr-xr-t 1 root bin 411089 Feb 3 1997 /opt/PGMS/bin/prgm
```

This also requires modifying the following tunable kernel parameter on the client:

```
page_text_to_local = 1
```

ca n'existe plus

There are a number of NFS tuning solutions that can help to improve performance on NFS clients:

- **Tune number of biod daemons:** The default number of **biod** daemons in HP-UX 11.00 and earlier was four. This most likely is too small. The best recommendation is to have a minimum of two **biod** daemons for every client process performing I/O to and from the NFS file system. Each **biod** daemon has, at most, one NFS request outstanding at any time, and as the number of **biod** daemons increase, the more disk requests the client can send. If the client has x process performing file system I/O and y **biod** daemons, then the client could have $x+y$ RPC requests outstanding at one time: one for each of the **biod** daemons, and one for each of the clients. In 11i, the default number of **biod** was raised to 16. This seems a more reasonable number.

The best indicator of too few **biod** daemons is the number of waits shown in the **netstat -c** command.

- **Tune the NFS Mount Options:** There are a number of NFS mount options that can affect client performance, among them the NFS read and write buffer sizes. The NFS buffer size (specified with the **rsize** and **wsizes** mount options) determines the increment in which data is transferred to and from the NFS file system. For example, if the file system block size is 8192 bytes, and the NFS buffer size was 8,500 bytes, two file system I/Os would be required before any NFS packet could be sent.

The recommendation for NFS buffer size is to match the size of the file system block size. The default NFS buffer size is 8192 bytes, and this does match the default file system block size on HFS. For JFS, try to match the buffer size to the size of a typical extent size.

11-18. SLIDE: Tuning the Network

Tuning the Network



Ways to Reduce LAN Congestions

- Subnet the network
- Use routers, not a computer for IP gateway
- Use higher speed LAN technology:
 - Fast Ethernet, FDDI, ATM
- Increase the number of LAN adapters on the server
- Put the server on an FDDI network and use routers to segment client traffic
- Put the server on an FDDI network and use switches to fan out to clients

de preference utiliser le montage hardware au lieu de software à cet effet il faut ajouter un tr

Student Notes

Subnetting a network is an effective way to reduce congestion on a LAN.

Using routers (as compared to bridges, Ethernet switches, and Ethernet to FDDI concentrators) provides a great deal of flexibility in the form of security, network segmentation, and routing of diverse types of packets. Routers usually provide good throughput and performance at a relatively low cost.

Using an existing computer system as a gateway for traffic between NFS clients and the file server is often inefficient and limits the performance of the NFS clients.

By making sure the maximum transmission unit MTU on the client system, the file server, and all routers in between them, the overhead on the routers caused by packet fragmentation and re-assembly can be avoided.

Module 11

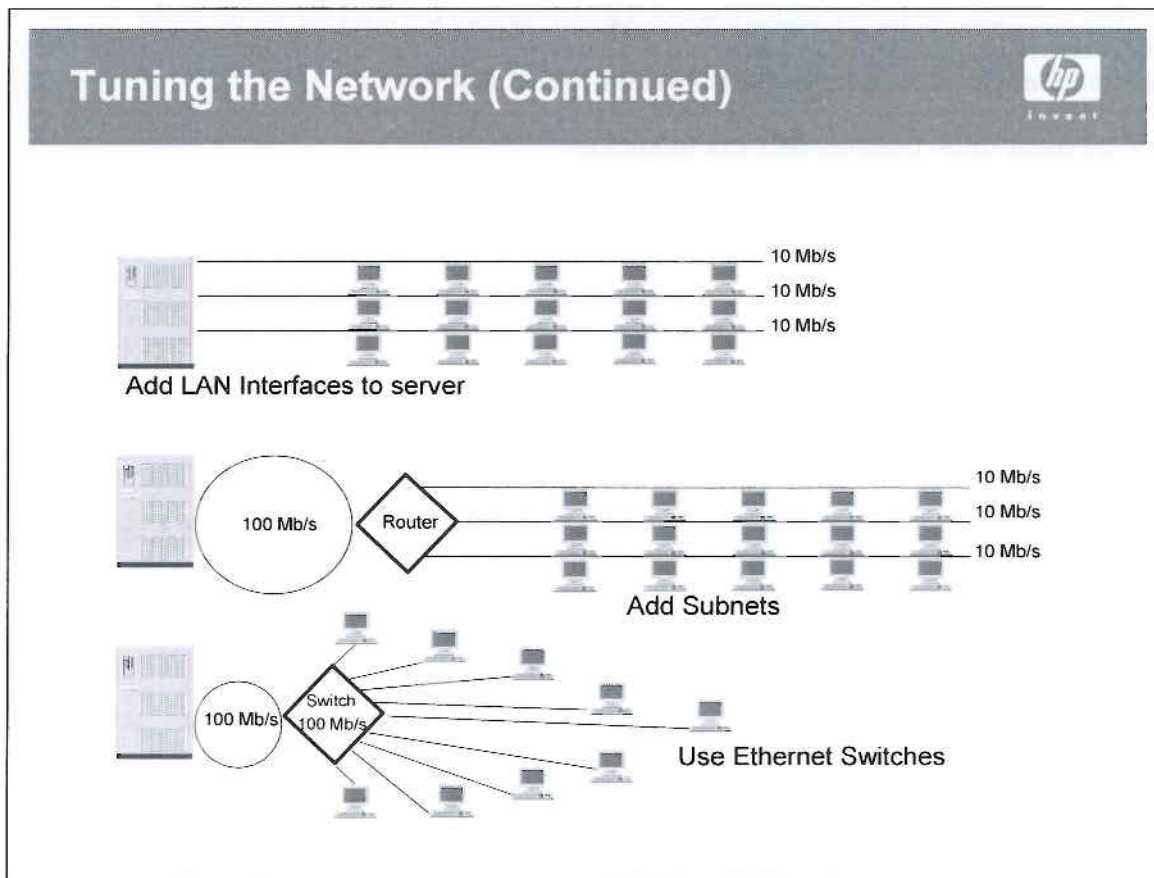
Network Performance

Example

```
# netstat -i
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
lo0    4608 loopback     localhost    6055   0       6055   0       0
lan0   1500 156.153.192 pr1w1        3724729 0      1705240 10      34739
lan1*  1500 none         none         0      0       0      0       0
```

The `route` command (`-p` option) can be used to set the `Path Mtu` size for a host route only.

11-19. SLIDE: Tuning the Network (Continued)



Student Notes

If the average client demand on an NFS server is measured to be greater than the network bandwidth, and assuming 100 clients demand 10 NFS requests per second, then a single 10-MB Ethernet segment (with a calculated 360 packets maximum per second) could not handle this workload, even though the server itself may be able to (from a processing stand point).

To allow this client workload to be processed by the single NFS server, the following network configurations can be implemented:

1. Use at least three network interface cards, one for each segment, distributing 33-34 clients on a segment.
2. Use one or more high-speed network connections, which connect to multiple lower bandwidth LAN segments.

In the first example, we have added multiple LAN interfaces to our NFS server.

In the second example, we have a 100-MB/second/ FDDI card on the NFS file server. We also have a router on the same segment as the server that has an FDDI interface, as well as

Module 11
Network Performance

several regular 10-MB/second Ethernet interfaces. Here, the issue of the routers ability to do packet fragmentation and reassembly efficiently, may become important.

In our last example, we have a 100-MB/second FDDI card on the NFS file server and a 100-MB/second translating Ethernet switch on the same FDDI segment. Since this is not routed, the file server and clients share the same subnet address.

There are many other possible network topologies.

11-20. LAB: Network Performance

Directions

The following two labs investigate Network read and write performance. The labs use NFS and are performed against the JFS file system created in the JFS module.

Lab 1 — Network Read Performance

To perform this lab, two systems are needed: an NFS server and an NFS client. Pair up with another student in the class for this lab.

1. Make sure the JFS file system on the NFS server contains the **make_files** program. Execute the **make_files** program to create files for the client to access.

```
# mount /dev/vg00/vxfs /vxfs
# cp /home/h4262*/disk/lab1/make_files /vxfs
# cd /vxfs
# ./make_files
```

2. Export the JFS file system so the client can mount it.

```
# exportfs -i -o root=client_hostname /vxfs
# exportfs
```

3. From the client system, mount the NFS file system.

```
# mount server_hostname:/vxfs /vxfs
```

4. Time how long it takes to read the 20 MB of files from the mounted file system. Record the results:

```
# timex cat /vxfs/file* > /dev/null
```

Record results:

Real: _____ User: _____ Sys: _____

5. Now that the data is in the client's buffer cache, time how long it takes to read the exact same files again. Record the results:

```
# timex cat /vxfs/file* > /dev/null
```

Record results:

Real: _____ User: _____ Sys: _____

Moral: Try to have a big enough buffer on the client system for a lot of data to be cached. Also, **biod** daemons will help prefetching data.

Module 11
Network Performance

6. Test to see if fewer `biod` daemons will change the initial performance.

```
# cd /
# umount /vxfs
# kill $(ps -e |grep biod|cut -c1-7)
# /usr/sbin/biod 4
# mount server_hostname:/vxfs /vxfs
# timex cat /vxfs/file* > /dev/null
```

Record results:

Real: _____ User: _____ Sys: _____

7. Once finished, remove the files and `umount` the file system.

```
# rm /vxfs/file*
# umount /vxfs
```


Lab 2 — Network Write Performance

The following lab has the client perform many writes to an NFS file system. The following parameters will be investigated:

- Number of **biod** daemons
- NFS version 2 versus NFS version 3
- TCP versus UDP

During this lab, the monitoring tools shown below should be used on the client and server

CLIENT	SERVER
# nfsstat -c	# nfsstat -s
# glance NFS report (n key)	# glance NFS report (n key)
# glance Global Process (g key)	# glance Global Process (g key)
- monitor biod daemons	-monitor nfsd daemons
# glance Disk report (d key)	
- monitor Remote Rds/Wrts	
-	

1. From the NFS client, mount the NFS file system as a version 2 file system.
mount -o vers=2 server_hostname:/vxfs /vxfs
2. Terminate all the **biod** daemons on the client.
kill \$(ps -e |grep biod|cut -c1-7)
3. Time how long it takes to copy the **vmunix** file to the mounted NFS file system. Record the results: The first command buffers the file.
cat /stand/vmunix >/dev/null
timex cp /stand/vmunix /jfs

Record results:

Real: _____ User: _____ Sys: _____

4. Now, start up the **biod** daemons, and retry timing the copy. Record the results:
/usr/sbin/biod 4
timex cp /stand/vmunix /jfs

Record results:

Real: _____ User: _____ Sys: _____

Module 11
Network Performance

5. Change the mount options to version 3 and retime the transfer:

```
# cd /
# umount /vxfs
# mount -o vers=3 server_hostname:/vxfs /vxfs
# cd /
# timex cp /stand/vmunix /vxfs
```

Record results:

Real: _____ User: _____ Sys: _____

6. Compare the speed of FTP to NFS. Transfer the file to the server using the `ftp` utility.

```
# ftp server_hostname
# put /stand/vmunix /vxfs/vmunix.ftp
```

How long did the FTP transfer take? _____

Explain the difference in performance.

7. Test the potential performance benefit of turning off the new TCP feature of HP-UX 11i. First, mount the file system with UDP protocol rather than the default TCP.

```
# umount /vxfs
# mount -o vers=3 -o proto=udp server_hostname:/vxfs /vxfs
```

Perform the copy test again and compare the results with the TCP version 3 mount data in part 3. Is UDP quicker than TCP?

```
# timex cp /stand/vmunix /vxfs
```

Module 12 — Tunable Kernel Parameters

Objectives

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

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

12-1. SLIDE: Kernel Parameter Classes

Kernel Parameters Classes



- Static
 - requires a kernel rebuild and a reboot
- Dynamic
 - changes take place immediately
 - changes survive a reboot
- Automatic
 - constantly being tuned by the kernel
 - can be set manually to a fixed value

y'ai la possibilité qu'un kernel dynamique devienne statique

Student Notes

There are a number of tunable parameters within the kernel that can have a big impact on performance. When making changes to these parameters, it may require that a new kernel be compiled. As of 11i v1, about 12 parameters were converted to dynamically tunable parameters. That is, their values could be changed without rebuilding the kernel and without rebooting the system. As of 11i v2, there are now around 36 dynamically tunable parameters, plus a few traditional parameters that are now tuned by the kernel, so no manual tuning of them need be done at all.

Static kernel parameters have been around since UNIX was first designed. In order to change one of these parameters, it was necessary to alter the contents of a system configuration file, **system**, rebuild the kernel using this altered configuration file, move the new kernel into place, and reboot the system to activate the new kernel. This tended to be time consuming and forced the system to become unavailable for a time.

Recently, with HP-UX 11i v1, a few kernel parameters were converted to dynamic tuning. These parameters could be altered, using SAM or **kmtune**, and the changes would become effective immediately. There was no longer a need to rebuild the kernel or reboot the system. However, this only applied to those few kernel parameters. The vast majority of kernel parameters were still static. The dozen parameters that were made dynamically tunable,

were ones that tended to be tuned by system administrators more frequently, but were relatively easy to convert to dynamic.

More recently, with HP-UX 11i v2, several more parameters were converted to dynamic tuning. These parameters were also tuned fairly frequently by system administrators, but were more difficult to convert to dynamic.

At the same time, a new class of parameters was introduced – automatic. These parameters were tuned by the kernel – constantly – in response to changing conditions in the system. However, the system administrator could override the automatic handling by the kernel and force the parameter to some fixed value, if needed.

At HP-UX 11i v1, the following kernel parameters became dynamic:

```
core_addshmem_read
core_addshmem_write
maxfiles_lim
maxtsiz
maxtsiz_64bit
maxuprc
msgmax
msgmnb
scsi_max_qdepth
semmsl
shmmax
shmseg
```

At HP-UX 11i v2, the following additional kernel parameters became dynamic:

```
aio_listio_max
aio_max_ops
aio_monitor_run_sec
aio_prio_delta_max
aio_proc_thread_pct
aio_proc_threads
aio_req_per_thread
alloc_fs_swapmap
alwaysdump
dbc_max_pct
dbc_min_pct
dontdump
fs_symlinks
ksi_alloc_max
max_acct_file_size
max_thread_proc
maxdsiz
maxdsiz_64bit
maxssiz
maxssiz_64bit
nfile
nflocks
nkthread
```

Module 12

Tunable Kernel Parameters

nproc
nsysmap
nsysmap64
physical_io_buffers
shmmni
vxfs_ifree_timelag

Also at HP-UX 11i v2, the following kernel parameters are obsolete or automatic:

bootspinlocks
clircreservedmem
maxswapchunks
maxusers
mesg
ncallout
netisr_priority
nni
ndilbuffers
sema
semmap
shmem
spread_UP_drivers

12-2. SLIDE: Tuning the Kernel

Tuning the Kernel



- Use `system_prep`, `kmtune`, or `kctune` to view current values of tunable kernel parameters.
- Use SAM (or new `km/kc` commands) to tune kernel parameters.
- Tune only one parameter at a time.
- Do not make parameters unnecessarily large.
- Use `glance` to monitor system table sizes (ensure highest value is not equal to total table size).
- Some kernel parameters are dynamic (no reboot) see `kmtune` and `kctune`.

en 11i v2 il y'a la possibilité de ~~de~~ faire des alarmes sur le kernel en envoyant des alarmes

Student Notes

Some general rules and notes regarding tuning and recompiling the kernel:

- View the existing, tunable parameters with the `kctune` command (HP-UX 11i v2), the `kmtune` command (HP-UX 11.00 and 11i v1) or the `sysdef` or `system_prep` commands (HP-UX 10.x). You can also use SAM with any version of HP-UX to view the current values. Examples of outputs are shown below.
- Use the System Administration Manager (SAM) to tune the kernel parameters and rebuild the systems. SAM has the advantage of displaying all available, tunable parameters, their current values, and a range of acceptable values. SAM also knows which parameters can be tuned dynamically and will make changes to them immediately. As of HP-UX 11i v2, SAM calls a separate utility to do the actual tuning.
- When tuning performance by modifying kernel parameters, modify only one value with each kernel rebuild. By changing several parameters at once, you may cloud the picture and make it much more difficult to determine what helped and what hurt the system's performance.

Module 12

Tunable Kernel Parameters

- Avoid setting the tunable parameters too large. Many of the parameters create in-core memory data structures whose size is dependent upon the value of the tunable parameter (for example, `nprocs` to the size of the process table). Generally, it is a good rule of thumb to increase or decrease a parameter by no more than 20%, while trying to find the best setting for it. Of course, if you are changing a parameter's value to accommodate some new application you are installing, always follow the manufacturer's suggested changes.
- Use `glance` to monitor system table sizes. Ensure the system tables are not running out of entries. In general, there should be around 20% of unused entries in any table. This will ensure that you have enough entries to handle any high demand periods.

The step-by-step procedure for tuning and recompiling the kernel manually on HP-UX 11.X is shown below:

1. Log in as superuser.
2. Change directory.

```
cd /stand/build
```
3. Create a system file from your current kernel.

```
/usr/sbin/sysadm/system_prep -v -s system
```
4. Modify the `/stand/build/system` file as desired.
5. Build the kernel: `/usr/sbin/mk_kernel -s system`.
6. Save your old system and kernel files, just in case you want to go back.

```
cp /stand/system /stand/system.prev  
cp /stand/vmunix /stand/vmunix.prev  
cp /stand/dlkm /stand/dlkm_vmunix.prev
```
7. Schedule the kernel update on the next reboot.

```
kmupdate
```
8. Shut down and reboot from your new kernel.

```
/sbin/shutdown -ry 0
```

Understanding Dynamic Kernel Variables.

`kctune(1M)`, `kmtune(1M)` or `sam` can be used “on the fly” to modify some kernel variables. Any changes take place immediately without the need to reboot. In HP-UX 11i v2, `kmtune` still exists, but simply calls `kctune`.

Example using kmtune to set and then activate a new value for a dynamic kernel variable.

```
# kmtune -q shmseg
Parameter      Current Dyn Planned  Module Version
-----
shmseg         120   Y   120

# kmtune -s shmseg=155

# kmtune -l -q shmseg
Parameter:      shmseg
Current:        120
Planned:        155
Default:        120
Minimum:        -
Module:         -
Version:        -
Dynamic:        Yes

# kmtune -u shmseg
shmseg has been set to 155 (0x9b).
```


12-3. SLIDE: Kernel Parameter Categories

Kernel Parameter Categories



- File system
- Message queues
- Semaphores
- Shared memory
- Process
- Swap
- LVM
- Networking
- Miscellaneous

Student Notes

The next few slides will present the tunable kernel parameters in these categories.

12-4. SLIDE: File System Kernel Parameters

File System Kernel Parameters			
Kernel Parameter	Default	Description	
<code>dbc_min_pct</code>	5	Minimum size of dynamic buffer cache (dbc)	
<code>dbc_max_pct</code>	50	Maximum size of dynamic buffer cache (dbc)	
<code>nbuf</code>	0	Number of buffer headers (in 10.x and above, use DBC)	
<code>bufpages</code>	0	Number of 4-KB buffer pages (in 10.x and above, use DBC)	
<code>fs_async</code>	0	If on (1), forces all meta-data writes to disk to be asynchronous	(pour renforcer l'écriture synchrone, ou asynchrone)
<code>maxfiles</code>	60	Soft limit to the number of files a process can have open	
<code>maxfiles_lim</code>	1024	Hard limit to the number of files a process can have open	
<code>nfile</code>	formula	Size of file table in memory	
<code>ninode</code>	formula	Size of inode table in memory	
<code>nflocks</code>	200	Size of file-lock table in memory	
<code>vx_ncsize</code>	1024	Size of vxfs directory name lookup cache (DNLC)	cache des pointeurs le clones q'itode

Student Notes

- dbc_min_pct** `dbc_min_pct` specifies the minimum size that the system's buffer cache may shrink to as a percentage of physical memory. It is now dynamic in 11i v2.
- dbc_max_pct** `dbc_max_pct` specifies the maximum size that the system's buffer cache may grow to as a percentage of physical memory. It is now dynamic in 11i v2.
- nbuf** `nbuf` is used to specify the number of file system buffer cache headers. Set `nbuf` to zero if you want to use the system's ability to grow and shrink this important table dynamically, based on demand. It is not yet obsolete, but expect it to be so in a future release.
- bufpages** `bufpages` specifies the number of 4-KB pages in memory that will be allocated for the file system buffer cache. Like `nbuf`, this parameter should be set to zero if you want to use the dynamic form of buffer cache allocation. If this value is non-zero, enough `nbufs` (one for every two `bufpages`) will be created as well, unless otherwise specified. It is not yet obsolete, but expect it to be so in a future release.

Module 12

Tunable Kernel Parameters

fs_async	fs_async specifies that file system data structures may be posted to disk asynchronously. While this can speed file system performance for some applications, it increases the risk that a file system will be corrupted in the event of system power loss.
maxfiles	maxfiles specifies the soft limit to the number of files that a single program may have open at one time. A program may exceed this soft limit up to the value of maxfiles_lim . In 11i v2, maxfiles is computed at boot and is set to 512, if memory is less than 1 GB. Otherwise it's set to 2048.
maxfiles_lim	maxfiles_lim is the hard limit to the number of files that a single program can open up at one time. This parameter was made dynamic in 11i v1 and the default value was set to 4096.
nfile	nfile is the size of the file table in memory, and therefore defines the maximum number of files that may be open at any one time on the system. Every process uses at least three file descriptors. Be generous with this number, as the required memory is minimal. nfile depends on the parameters nproc , maxusers , and npty . This parameter was made dynamic in 11i v2 and was no longer dependent on maxusers . Its value is computed at boot time and is set to 16384 if memory is less than 1 GB; otherwise it's set to 65536.
ninode	ninode is the size of the HFS in-core inode table. By caching inodes in memory the amount of physical I/O is decreased when accessing files. Each unique HFS file open on the system has a unique inode . This table is hashed for performance. At boot time in 11i v2, it's set to 4880, if memory is less than 1GB; otherwise it's set to 8196.
nflocks	nflocks is the number of file locks available on the system. File locks are a kernel service to enable applications to safely share files. Databases or other applications that make use of the lockf() system call can be large consumers of file locks. Note that one file may have several locks associated with it. This parameter was made dynamic in 11i v2 - at boot time, if memory is less than 1 GB, it's set to 1200; otherwise it's set to 4096.
vx_ncsize	Along with ninode , this parameter controls the size of the DNLC (directory name lookup cache). Recent directory path names are stored in memory to improve performance. This parameter is set in bytes. This parameter has been obsoleted in 11i v2. VxFS 3.5 now uses its own internal DNLC.

12-5. SLIDE: Message Queue Kernel Parameters

Message Queue Kernel Parameters 		
Kernel Parameter	Default	Description
<code>mesg</code>	1	Enable or disable IPC messaging (700 only)
<code>msgmap</code>	formula	Size of message-free-space map
<code>msgmax</code>	8192	Maximum size in bytes of an individual message
<code>msgmnb</code>	16384	Maximum size in bytes of message queue space
<code>msgmni</code>	50	Maximum number of messages queue identifiers
<code>msgseg</code>	2048	Number of segments in the the system message buffer
<code>msgsz</code>	8	Size in bytes of segments to be allocated for messages
<code>msgtql</code>	40	Size of message header space (1 header per message)

Student Notes

Message queues are used by applications to transfer a small to medium amount of information from exactly one process to another process. This information could be in the form of a structure, a string, a numerical value, or any combination thereof. SVIPC message queues have been around for a long time. They are controlled by a number of tunable kernel parameters.

<code>mesg</code>	<code>mesg</code> when set (<code>mesg = 1</code>) enables the message queue services in the kernel. This parameter is obsolete as of 11i v2.
<code>msgmap</code>	<code>msgmap</code> specifies the size of the free-space map used in allocating message buffer segments for messages.
<code>msgmax</code>	<code>msgmax</code> specifies the maximum size in bytes of an individual message. This parameter is dynamic at HP-UX 11i v1.
<code>msgmnb</code>	<code>msgmnb</code> specifies the maximum total space consumed by all messages in a queue. This parameter is dynamic at HP-UX 11i v1.
<code>msgmni</code>	<code>msgmni</code> specifies the maximum number of message queue identifiers allowed on the system at one time. Each message queue has an associated message

Module 12

Tunable Kernel Parameters

	queue identifier stored in non-swappable kernel memory. In 11i v2, the default was raised to 512.
msgseg	msgseg is the number of segments in the system-wide message buffer. In 11i v2, the default was raised to 8192.
msgssz	msgssz is the size in bytes of each message buffer segment. In 11i v2, the default was raised to 96.
msgtql	msgtql is the total number of messages that can reside on the system at any on time. In 11i v2, the default was raised to 1024.

Any of these parameters could affect the performance of an application, simply by virtue of not having enough of the message queue resources available when needed. However, the **msgssz** and the **msgseg** parameters also control the size in an in-memory message buffer that is shared by all SVIPC message queues. It needs to be large enough to handle all the messages that may be pending at any one time, but by the same token, should not be much larger than that. It could be taking up far more memory than is necessary. It is not dynamic; it is fixed in size.

There also exist in HP-UX 11.x POSIX message queues. There are no tunable parameters for them. POSIX message queues have been shown to consistently out-perform SVIPC message queues.

12-6. SLIDE: Semaphore Kernel Parameters

Semaphore Kernel Parameters		
Kernel Parameter	Default	Description
<code>sema</code>	1	Enable or disable Semaphore code (700 only)
<code>semaem</code>	16384	Maximum amount a semaphore can be changed by undo
<code>semmap</code>	formula	Size of free-space map used for allocating new semaphores
<code>semmni</code>	64	Maximum number of sets of semaphores
<code>semmns</code>	128	Maximum number of semaphores, system-wide
<code>semmnu</code>	30	Maximum number of processes that can have undo operations pending on a given semaphore
<code>semume</code>	10	Maximum number of semaphores that a given process can have undo operations pending on
<code>semvmx</code>	32767	Maximum value a semaphore is allowed to reach
<code>semmsl</code>	2048	Maximum number of semaphores in a given set

Student Notes

Semaphores are another form of interprocess communication. Semaphores are used mainly to keep processes properly synchronized to prevent collisions when accessing shared data structures.

Semaphores are typically incremented or decremented by a process to block other processes while it is performing a critical operation or using a shared resource. When finished, it decrements or increments the value, allowing blocked processes to then access the resource. Semaphores can be configured as binary semaphores with only two values: 0 and 1, or they can serve as general semaphores (or counters), where one process increments/decrements the semaphore and one or more cooperating processes decrement/increment it.

SVIPC semaphores have been around for a long time. They are controlled by several tunable parameters.

- sema** **sema** (Series 700 only) enables or disables IPC semaphores at system boot time. This parameter is obsolete as of 11i v2.
- semaem** **semaem** is the maximum value by which a semaphore can be changed in a semaphore undo operation.

Module 12

Tunable Kernel Parameters

semmap	semmap is the size of the free-semaphores resource map for allocating requested sets of semaphores. This semaphore is obsolete as of 11i v2.
semnmi	semnmi is the maximum number of sets of IPC semaphores allowed on the system at any given time. In 11i v2, the default was raised to 2048.
semnms	semnms is the total system-wide number of individual IPC semaphores available to system users. In 11i v2, the default was raised to 4096.
semnmu	semnmu is the maximum number of processes that can have undo operations pending on any given IPC semaphore on the system. In 11i v2, the default was raised to 256.
semume	semume is the maximum number of IPC semaphores on which a given process can have undo operations pending. In 11i v2, the default was raised to 100.
semvmx	semvmx , the maximum value any given IPC semaphore is allowed to reach, prevents undetected overflow conditions).
semmsl	Until 11i v2, semmsl was an untunable value in the kernel. It specified the maximum number of semaphores that could be allocated to a specific semaphore set. In 10.X it was set to 500. In 11.00, it was set to 2048. Now it is a dynamic tunable.

Any of these parameters could affect the performance of an application, simply by virtue of not having enough of semaphore resources available when needed.

There also exist in HP-UX 11.x POSIX semaphores. There are no tunable parameters for them. POSIX semaphores have been shown to consistently out-perform SVIPC semaphores.

12-7. SLIDE: Shared Memory Kernel Parameters

Shared Memory Kernel Parameters		
Kernel Parameter	Default	Description
<code>shmem</code>	1	Enable or disable Shared Memory (700 only)
<code>shmmax</code>	64 MB	Maximum shared memory segment size
<code>shmuni</code>	200	Maximum number of total shared memory segments
<code>shmseg</code>	120	Maximum number of shared memory segments that a single process may attach

Student Notes

Shared memory is reserved memory space for storing data shared between or among cooperating processes. Sharing a common memory space eliminates the need for copying or moving data to a separate location before it can be used by other processes, reducing processor time and overhead, as well as memory consumption.

Shared memory is allocated in swappable, shared memory space. Data structures for managing shared memory are located in the kernel. Shared memory segments are much preferred by memory intensive applications, such as Data Bases, since they can be very large and can be accessed without using system calls. SVIPC shared memory use the following tunable parameters.

<code>shmem</code>	<code>shmem</code> , when set to true, enables the shared memory subsystem at boot time. This parameter is obsolete in 11i v2.
<code>shmmax</code>	<code>shmmax</code> specifies the maximum shared memory segment size. Dynamic in 11i v1. Also in 11i v2, the default was raised to 1GB.
<code>shmuni</code>	<code>shmuni</code> specifies the maximum number of shared memory segments allowed on the system at any one time. Dynamic in 11i v2. Also in 11i v2, the default was raised to 400.

shmseg **shmseg** specifies the maximum number of shared memory segments that can be simultaneously attached (**shmat()**) to a single process. Dynamic in 11i v1. Also in 11i v2, the default was raised to 300.

Any of these parameters could affect the performance of an application, simply by virtue of not having enough shared memory resources available when needed.

There also exist in HP-UX 11.x POSIX shared memory. There are no tunable parameters for them. POSIX shared memory segments are implemented through the memory-mapped file architecture, so it could be affected by some of the file system tunable parameters described earlier.

12-8. SLIDE: Process-Related Kernel Parameters

Process-Related Kernel Parameters 		
Kernel Parameter	Default	Description
<code>maxdsiz</code> <code>maxdsiz_64bit</code>	256 MB	Maximum 32 and 64 bit process data segment size
<code>maxssiz</code> <code>maxssiz_64bit</code>	8 MB	Maximum 32 and 64 bit process stack size
<code>maxtsiz</code> <code>maxtsiz_64bit</code>	64 MB	Maximum 32 and 64 bit process text segment size
<code>maxressiz</code> <code>maxressiz_64bit</code>	8 MB	Maximum 32 and 64 bit process RSE stack size (IA-64 only)
<code>maxuprc</code>	50	Maximum number of concurrent processes per user ID
<code>nproc</code>	formula	Maximum number of processes system wide
<code>timeslice</code>	8	Maximum time a process can have the CPU before yielding to next highest priority. Set in "ticks" (10ms).

Student Notes

Manage the number of processes on the system and processes per user to keep system resources effectively distributed among users for optimal overall system operation.

Manage allocation of CPU time to competing processes at equal and different priority levels.

Allocate virtual memory among processes, protecting the system and competing users against unreasonable demands of abusive or run-away processes.

maxdsiz **maxdsiz** defines the maximum size of the static data storage segment of an executing 32-bit process. In 11i v2, this default has been raised to 1 GB.

maxdsiz_64bit **maxdsiz_64bit** defines the maximum size of the static data storage segment of an executing 64-bit process. In 11i v2, this default has been raised to 4 GB.

maxssiz **maxssiz** defines the maximum size of the dynamic storage segment (DSS), also called the stack segment, of an executing 32-bit process.

Module 12
Tunable Kernel Parameters

maxssiz_64bit	maxssiz_64bit defines the maximum size of the dynamic storage segment (DSS), also called the stack segment, of an executing 64-bit process. In 11i v2, this default has been raised to 256 MB.
maxtsiz	maxtsiz defines the maximum size of the shared text segment (program storage space) of an executing process. Note maxtsiz_64bit for 64 bit HP-UX 11.
maxressiz	maxressiz defines the maximum size of the register stack engine (RSE), also called the RSE stack segment, of an executing 32-bit process. This parameter is only found on an IA-64 kernel.
maxressiz_64bit	maxressiz_64bit defines the maximum size of the register stack engine (RSE), also called the RSE stack segment, of an executing 64-bit process. This parameter is only found on an IA-64 kernel.
maxuprc	maxuprc establishes the maximum number of simultaneous processes available to each user on the system. The user ID number identifies a user. The <i>superuser</i> is immune to this limit. In 11i v2, this default is now set to 256.
nproc	nproc specifies the maximum total number of processes that can exist simultaneously in the system. This parameter has been made dynamic in 11i v2, and the new default setting is 4200.
timeslice	The timeslice interval is the amount of time one thread is allowed to accumulate before the CPU is given to the next thread at the same priority. The value of timeslice is specified in units of (10 millisecond) clock ticks.

12-9. SLIDE: Memory-Related Kernel Parameters

Memory-Related Kernel Parameters 		
Kernel Parameter	Default	Description
<code>vps_ceiling</code>	16	Maximum automatic page size (kbytes) the kernel selects
<code>vps_chattr_ceiling</code>	1048576	Maximum page size (kbytes) useable with <code>chattr</code>
<code>vps_pagesize</code>	4	Default page size used without <code>chattr</code> specification
<code>swapmem_on</code>	1	Enable or disable pseudo swap
<code>nswapdev</code>	10	Maximum number of device swap areas
<code>nswapfs</code>	10	Maximum number of file system swap areas
<code>swchunk</code>	2048	Size in DEV_BSIZE (1-KB) units of swap space units
<code>maxswapchunks</code>	256	Maximum number of swchunk units
<code>page_text_to_local</code>	0	Enable or disable process text to be swapped locally <i>disable to swap text (bunnie)</i>

Student Notes

Configurable kernel parameters for memory paging enforce operating rules and limits related to virtual memory (swap space).

`vps_ceiling` This parameter is provided as a means to minimize lost cycle time caused by TLB (translation look-aside buffer) misses on systems using newer PA-RISC devices such as the PA-8000 and the Itanium family that have smaller TLBs and may not have a hardware TLB walker.

If a user application does not use the `chattr` command to specify a page size for program text and data segments, the kernel selects a page size that, based on system configuration and object size, appears to be suitable. This is called transparent selection.

`vps_chattr_ceiling` User applications can use the `chattr` command to specify a page size for program text and data segments, providing some flexibility for improving overall performance, depending on system configuration and object size. The specified size is then

	compared to the page-size value limit defined by <code>vps_chattr_ceiling</code> that is defined in the kernel at system-boot time. If the value specified is larger than <code>vps_chattr_ceiling</code> , <code>vps_chattr_ceiling</code> is used.
<code>vps_page_size</code>	Specifies the default user-page size (in Kbytes) that is used by the kernel if the user application does not use the <code>chattr</code> command to specify a page size.
<code>swapmem_on</code>	<code>swapmem_on</code> enables or disables the creation of pseudo-swap, which is swap space designed to increase the apparent total swap space, so that real swap can be used completely, or large memory systems don't need corresponding swap space.
<code>nswapdev</code>	<code>nswapdev</code> specifies an integer value equal to the number of physical disk devices that can be configured for device swap up to the maximum limit of 25.
<code>nwapfs</code>	<code>nwapfs</code> specifies an integer value equal to the number of file systems that can be made available for file-system swap, up to the maximum limit of 25.
<code>swchunk</code>	<code>swchunk</code> defines the chunk size for swap. This value must be an integer power of two. When the system needs swap space, one swap chunk is obtained from a device or file system. When that chunk has been used and another is needed, a new chunk is obtained. If the swap space is full or if there is another swap space at the same priority, the new chunk is taken from a different device or file system, thus distributing swap use over several devices.
<code>maxswapchunks</code>	<code>maxswapchunks</code> specifies the maximum amount of configurable swap space on the system. In 11i v2 this parameter is obsolete.
<code>page_text_to_local</code>	<code>page_text_to_local</code> allows NFS clients to write the text segment to local swap and retrieve it later. This eliminates two separate text-segment data transfers to and from the NFS server, thus improving NFS client program performance. This parameter does not seem to be defined in 11i v2, even though it has not been identified as an obsolete parameter.

12-10. SLIDE: LVM-Related Kernel Parameters

LVM-Related Kernel Parameters		
Kernel Parameter	Default	Description
<code>maxvgs</code>	10	Maximum number of volume group on the system
<code>no_lvm_disks</code>	0	Enable or disable system to use LVM (0 = false, LVM exists 1 = true, no LVM disks exist)

↓ default 100

Student Notes

Two configurable kernel parameters are provided that relate to kernel interaction with the logical volume manager.

maxvgs **maxvgs** defines the maximum number of volume groups configured by the logical volume manager on the system.

no_lvm_disks **no_lvm_disks** flag notifies the kernel when no logical volumes exist on the system, i.e. LVM is disabled. This parameter does not seem to be defined in 11i v2, although it is not identified as an obsolete parameter.

12-11. SLIDE: Networking-Related Kernel Parameters

Networking-Related Kernel Parameters		
Kernel Parameter	Default	Description
<code>netisr_priority</code>	1	Define priority to assign to the network packet processing daemon (-1 means handle on an interrupt basis – best packet processing performance)
<code>netmemmax</code>	10% of mem	Amount of memory, in bytes, to be allocated for IP packet fragmentation reassembly queue

Student Notes

Two configurable kernel parameter are related to the kernel's interaction with the networking subsystems:

netisr_priority **netisr_priority** sets the real-time interrupt priority for the networking interrupt service routine daemon. By default, it is set to 1 on Uniprocessor systems and 100 on Multiprocessor systems. This parameter is obsolete in 11i v2.

netmemmax `netmemmax` specifies how much memory is reserved for use by networking for holding partial Internet protocol (IP) messages which are typically held in memory for up to 30 seconds. When messages are transmitted using Internet protocol, they are sometimes broken into multiple, "partial" messages (fragments). `netmemmax` simply establishes a maximum amount of memory that can be used for storing network-message fragments until they are reassembled. This parameter does not seem to be defined in 11i v2, although it is not identified as an obsolete parameter.

12-12. SLIDE: Miscellaneous Kernel Parameters

Miscellaneous Kernel Parameters 		
Kernel Parameter	Default	Description
<code>create_fastlinks</code>	0	Enable or disable creation of fast symbolic links
<code>default_disk_ir</code>	1	Enable or disable immediate reporting on all disks
<code>maxusers</code>	32	Maximum number of simultaneous users expected
<code>ncallout</code>	formula	Maximum number of timeouts (for example, alarms) pending
<code>npty</code>	60	Maximum number of concurrent pseudo tty connections
<code>rtsched_numpri</code>	32	Number of distinct POSIX real-time priorities
<code>unlockable_mem</code>	0	Minimum amount of memory to reserved for use by the paging system

Student Notes

The following parameters are more or less unrelated.

<code>create_fastlinks</code>	When <code>create_fastlinks</code> is non-zero, it causes the system to create HFS symbolic links in a manner that reduces the number of disk-block accesses by one for each symbolic link in a pathname lookup.
<code>default_disk_ir</code>	<code>default_disk_ir</code> enables or disables immediate reporting. With Immediate Reporting ON, disk drives that have data caches return from a <code>write()</code> system call when the data is cached, rather than returning after the data is written on the media. This sometimes enhances write performance, especially for sequential transfers. In 11i v2, this parameter is set to 0, by default.
<code>maxusers</code>	<code>maxusers</code> does not itself determine the size of any structures in the system; instead, the default value of other global system parameters depends on the value of <code>maxusers</code> . When other configurable parameter values are defined in terms of <code>maxusers</code> , the kernel is made smaller and more efficient by minimizing wasted space due to improperly balanced resource allocations. In

Module 12
Tunable Kernel Parameters

	11i v2, the use of <code>maxusers</code> has been eliminated from the formula of every parameter that was dependent on it. Changing its value has no effect on 11i v2.
<code>ncallout</code>	<code>ncallout</code> specifies the maximum number of timeouts that can be scheduled by the kernel at any given time. A general rule is that one callout per process should be allowed unless you have processes that use multiple <code>callouts</code> . In 11i v2 this parameter is obsolete.
<code>npty</code>	<code>npty</code> specifies the maximum number of pseudo-tty data structures available on the system.
<code>rtsched_numpri</code>	<code>rtsched_numpri</code> specifies the number of distinct priorities that can be set for POSIX real-time processes running under the real-time scheduler.
<code>unlockable_mem</code>	<code>unlockable_mem</code> defines the minimum amount of memory that always remains available for virtual memory management and system overhead.

- un software qui existe sur internet qui s'appelle sim
System insight manager freeware il contient
serveur Igmt, Sam, Depot
Hp System Insight manager

Module 13 — Putting It All Together

Objectives

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

- 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.

13-1. SLIDE: Review of Bottleneck Characteristics

Review of Bottleneck Characteristics		
CPU	Disk	Memory
• High CPU utilization	• High CPU utilization • High disk utilization	• High CPU utilization • High disk utilization • High memory utilization (with swapping)

Student Notes

The above slide recaps the characteristics related to the three main performance bottlenecks.

CPU Bottlenecks

CPU bottlenecks often exhibit the following characteristics:

- High CPU usage due to lots of processes competing for the CPU
Large number of processes in the CPU run queue
- No disk bottleneck problems; disk utilization is low, few to no I/O requests in the disk queues
- No memory bottleneck problems; **vhand** not needing much, no paging to swap devices

Disk Bottlenecks

Disk bottlenecks often exhibit the following characteristics:

- High CPU usage due to the disk device drivers constantly executing to perform the I/O and user/system processes continually running to submit the I/O requests

- High disk utilization due to lots of I/O requests being continually submitted.
- No memory bottleneck problems; **vmstat** not needing much, no paging to swap devices

Memory Bottlenecks

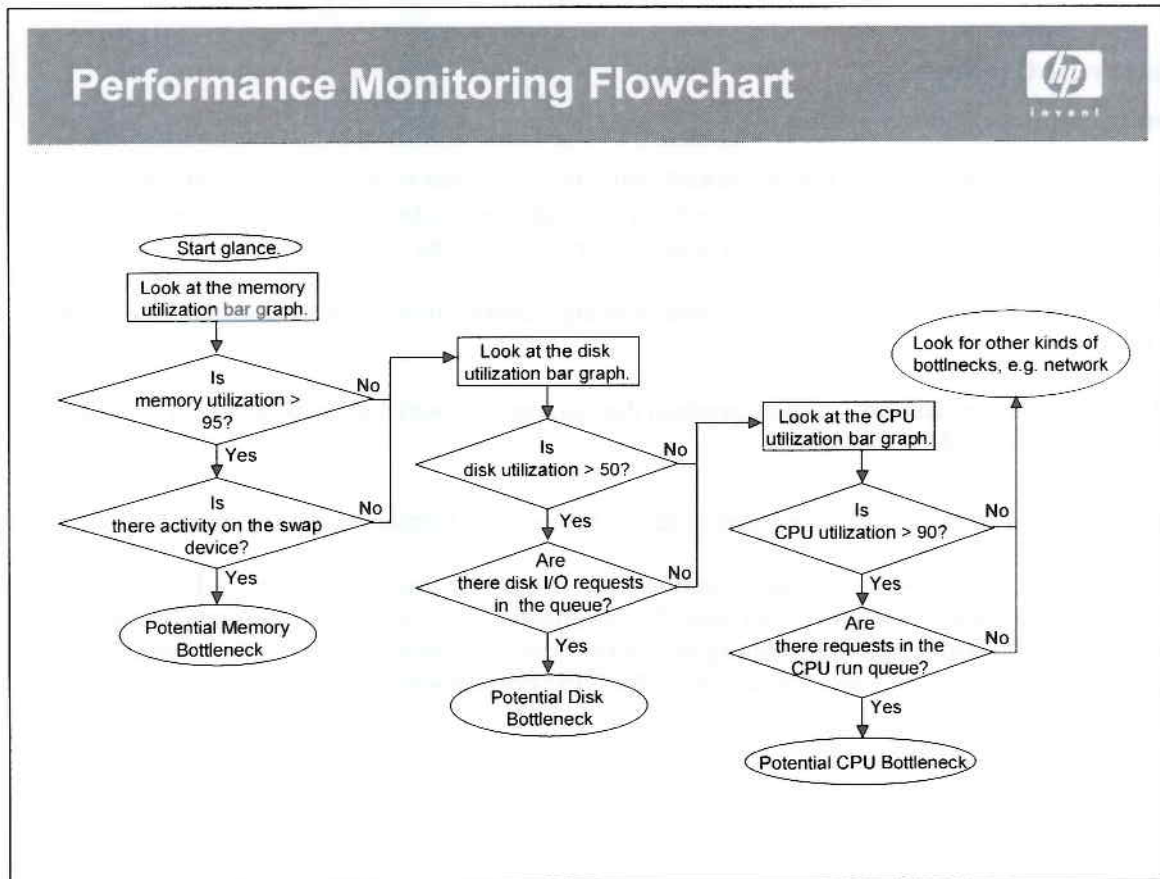
Memory bottlenecks often exhibit the following characteristics:

- High CPU usage (system) due to **vmstat** constantly running to free memory pages, the kernel spending lots of time in the memory management subsystem, and the device drivers for the disk writing memory pages to and from swap
- High disk utilization due to memory pages being constantly written to and from the swap devices
- High memory utilization (with swapping) due to free memory falling below **LOTSFREE**, **DESFREE**, and **MINFREE**

Given the above recap, in what order should the three main bottlenecks be checked?

When arriving on the scene of an unknown system, where do you start? It would be wise to look for the bottleneck with the most specific symptoms, first. Since the memory bottleneck is the only one to show signs of memory pressure, look for it first. Once you have eliminated that, look for disk bottlenecks. Finally, look for CPU bottlenecks.

13-2. SLIDE: Performance Monitoring Flowchart



Student Notes

The above performance monitoring flow chart assumes **glance** is being used as the performance-monitoring tool. If **glance** is not available, the same information can be obtained from a variety of other tools, such as **sar** and **vmstat**.

The flow chart starts by first looking for symptoms of a memory bottleneck.

- Is memory utilization high?
- Is there activity to the swap device?

Memory bottlenecks are checked for first, since memory bottlenecks often exhibit symptoms of high disk and CPU utilization, which could initially be mistaken for disk or CPU bottlenecks.

If the system is not bottlenecked on memory, the second bottleneck checked for through the flow chart is a disk bottleneck.

- Is disk utilization high?
- Are there disk I/O requests in the disk queue?

Disk bottlenecks are checked for second, as disk bottlenecks often exhibit symptoms of high CPU utilization, but not high memory utilization.

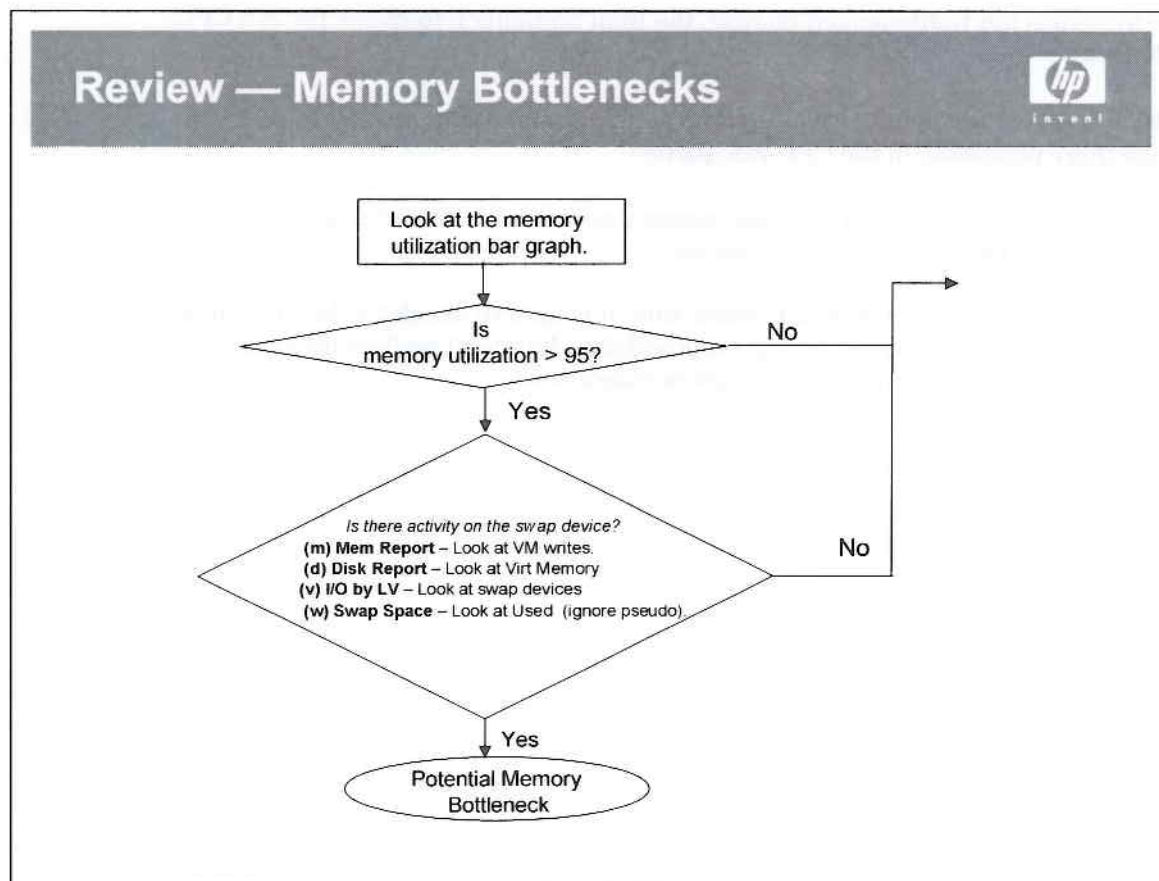
If the system is not bottlenecked on disk, the final bottleneck to check for is a CPU bottleneck.

- Is CPU utilization high?
- Are there processes in the CPU run queue?

CPU bottlenecks are checked for after memory and disk bottlenecks, as CPU bottlenecks do not exhibit high memory or CPU utilization.

If none of these situations appear to exist, then it is time to check the less common bottlenecks. Networks would be a good possibility, but don't neglect other hardware or even software resources, such as file locks and semaphores.

13-3. SLIDE: Review — Memory Bottlenecks



Student Notes

The primary symptoms of a memory bottleneck include *high memory utilization* and *activity to the swap device*.

The **glance** reports that show activity on the swap device include:

- (m) Memory Report - shows currently number of VM reads/writes
- (d) Disk Report - shows VirtMem I/O
- (v) I/O by log. volume - shows I/O to the swap logical volumes
- (w) Swap Space Report - show currently used swap space

Also look at **vhand** and **swapper** as processes. Are they accumulating any CPU time? Look at the output of **vmstat -s**. Are pages being paged out? Are processes being swapped out?

13-4. SLIDE: Correcting Memory Bottlenecks

Correcting Memory Bottlenecks



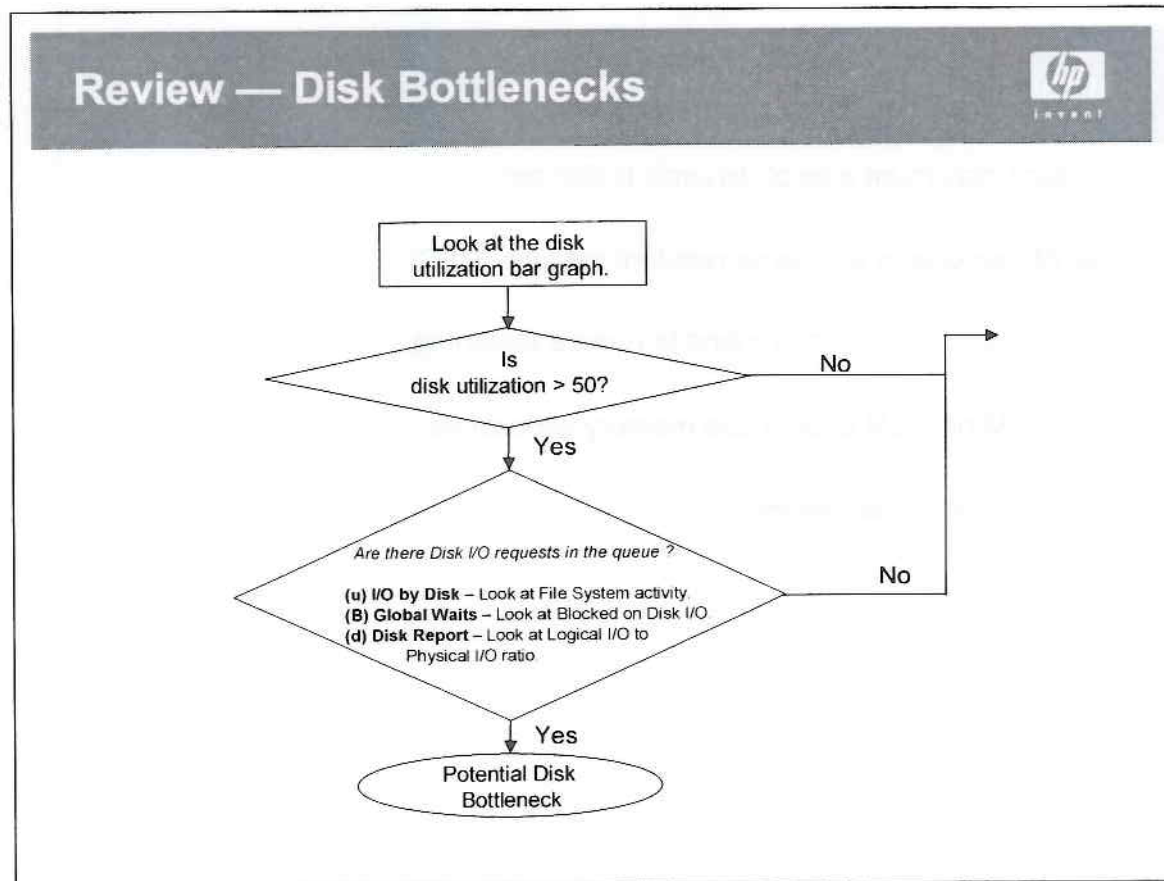
- Reduce maximum size of dynamic buffer cache.
- Identify programs with large resident set size (RSS).
- Use the `serialize` command to reduce thrashing.
- Use PRM or WLM to prioritize memory allocations.
- Add more physical memory.

Student Notes

The above slide reviews some of the ways to correct a memory bottleneck:

- Limit the maximum size of the dynamic buffer cache. This can help to prevent unnecessary paging during periods when the dynamic buffer cache needs to shrink.
- Identify programs (and users) taking up large amounts of memory, and investigate whether the memory usage is warranted or whether the process has memory leaks.
- Consider using the `serialize` command to keep several memory intensive programs from competing with each other.
- Consider using the Process Resource Manager (PRM) or Work Load Manager (WLM) to favor memory allocation to important processes.
- Adding more physical memory will always help a memory-constrained system.

13-5. SLIDE: Review — Disk Bottlenecks



Student Notes

The primary symptoms of a disk bottleneck include *high disk utilization* and *multiple I/O requests in the disk queue*.

The **glance** reports that show disk I/O related activity include:

- (u) I/O by Phys. Disk - shows currently number of reads/writes
- (B) Global Waits - shows percentage of processes blocked on Disk I/O
- (d) Disk Report - shows Logical I/O and Physical I/O activity

Also check the output of **sar -u (%wio)**, **sar -d**, and **sar -b** (for read cache hit rate and write cache hit rate).

13-6. SLIDE: Correcting Disk Bottlenecks

Correcting Disk Bottlenecks



- Load balance across disk drives and disk controllers.
- Consider asynchronous instead of synchronous I/O.
- Tune file system block and fragment/extent size.
- Tune file system (`vxfs` and `hfs`) mount options.
- Tune `vxfs` file systems with `vxtunefs`.
- Tune buffer cache for better hit ratios.
- Add additional and faster disk drives and controllers.

Student Notes

The above slide reviews some of the ways to correct a disk bottleneck:

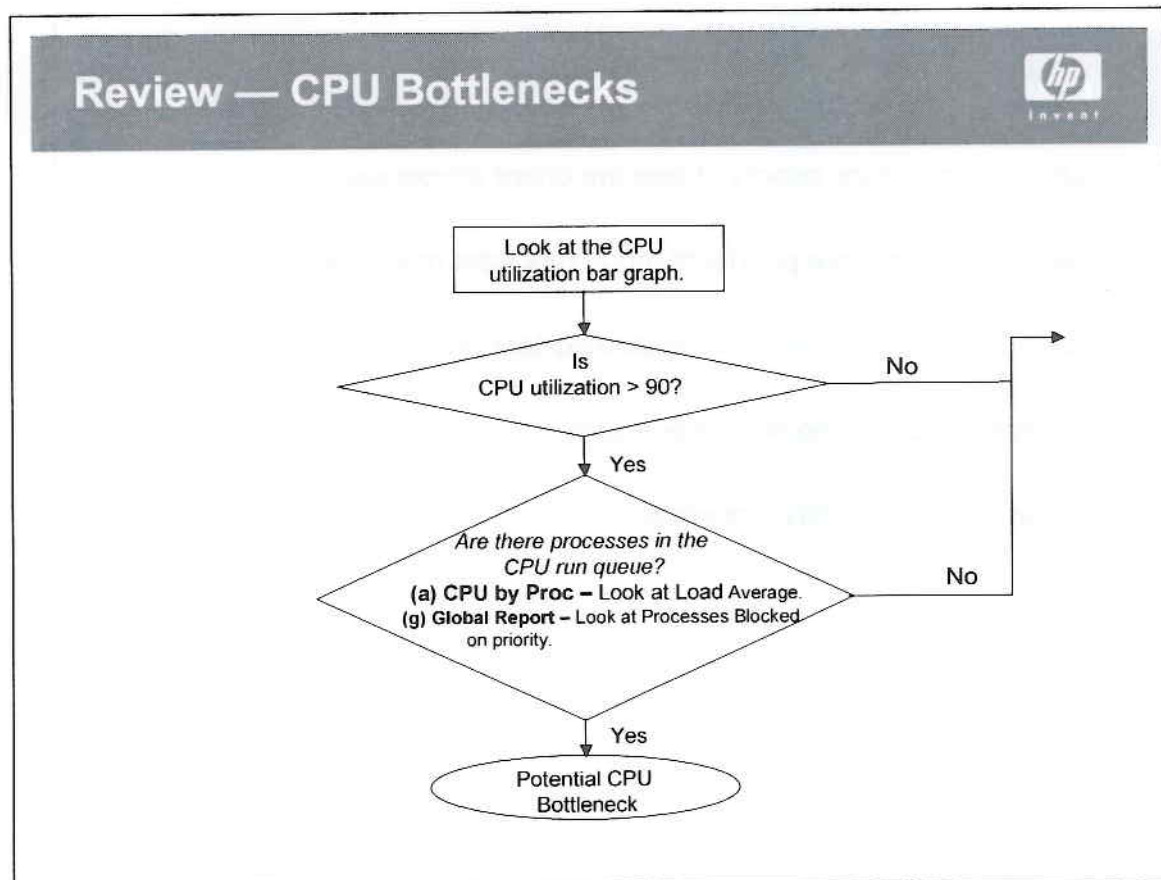
- Spread the I/O activity, as evenly as possible, over the disk drives and disk controllers.
- Consider using asynchronous I/O so applications do not have to wait for a physical I/O to complete. The trade-off here is a greater exposure to data loss in the event of a system failure.
- For HFS file systems, increase the fragment and file system block size if large files are being accessed in a sequential manner. For VxFS file systems, increase the block size to improve read-ahead and write-behind. Consider using a fixed extent size.
- Look at customizing file system mount options (especially for VxFS file systems). Recall that, by default, VxFS is mounted to favor integrity, and HFS is mounted to favor performance.
- Consider using `vxtunefs` to tune the performance of VxFS. Match preferred IO size and read ahead to physical stripe depth.

Module 13

Putting It All Together

- Verify (and tune) the hit ratio on the file system buffer cache. The ratio of logical reads to physical reads should be a minimum of 10 to 1. The ratio of logical writes to physical writes should be a minimum of 3 to 1.
- Add bigger, better, faster disks and disk controllers.

13-7. SLIDE: Review — CPU Bottlenecks



Student Notes

The primary symptoms of a CPU bottleneck include *high CPU utilization* and *multiple processes in the CPU run queue*.

The **glance** reports that show CPU activity include:

- (a) CPU by Processor - shows CPU load average over last 1, 5, 15 minutes
- (c) CPU Report - shows CPU activities
- (g) Process Report - shows CPU hogs in order (see note)

Note	Make sure you are looking at processes in CPU order. Use the Thresholds Page (o) of glance and set "CPU" as the sort criteria.
------	---

Also check **sar -u** and **sar -q**. Use the **-M** option, if you have a multiprocessor.

13-8. SLIDE: Correcting CPU Bottlenecks

Correcting CPU Bottlenecks



- Use `nice` to reduce priority of less important processes.
- Use `nice` to improve priority of more important processes.
- Use `rtprio` or `rtsched` on most important processes.
- Run batch jobs during non-peak hours.
- Add another (or faster) processor.

Student Notes

The above slide reviews some of the ways to correct a CPU bottleneck:

- Use the `nice` or `renice` commands on lower priority processes (set `nice` value to 21-39). As a rule of thumb, favor I/O bound programs over CPU-bound programs. I/O-bound programs will block frequently, allowing the CPU-bound programs to run.
- Use the `nice` or `renice` command on higher priority processes (set `nice` value to 0-19).
- Use the `rtprio` or `rtsched` commands on highest priority processes. BE CAREFUL! A poorly written process could take over your system and render it useless.
- Schedule large batch jobs, long compiles, and other CPU intensive activity for non-peak hours.
- Add an additional CPU or a faster CPU to the system.

13-9. SLIDE: Final Review — Major Symptoms

Final Review – Major Symptoms



Memory Bottleneck:	Both vhand and swapper active
Disk Bottleneck:	Disk utilization > 50% Request queues > 3
CPU Bottleneck:	CPU utilization > 90% Run queues > 3 per processor
Network Bottleneck:	Collisions/out-bound packets > 5%

All conditions sustained over time!

Student Notes

Let's summarize the major bottlenecks and their symptoms:

Memory Bottleneck: You know that you have a memory bottleneck if both **vhand** and **swapper** are active. This indicates severe memory pressure!

Disk Bottleneck: A disk bottleneck will be characterized by disk utilization of at least 50% and at least 3 requests waiting in the request queue. If a controller is the bottleneck, you will see multiple disks with lengthy queues on that controller. Their utilization may not be 50%! The queues are more important than the utilization.

CPU Bottleneck: If all of your CPUs are at least 90% busy and they each have run queues that have 3 or more processes in them, you have a CPU bottleneck. If one or more of the processors has empty (or mostly empty) queues, either you are at the limit of your CPU resource, or something is unbalancing the loads on your processors.

Network Bottleneck: If your ratio between your collisions/sec and your packets-out/sec is greater than 5%, you have a network bottleneck.

Module 13

Putting It All Together

As with any bottleneck symptom, it must be a constant condition – sustained over time to be considered a true bottleneck. Otherwise, it's a momentary spike which we will keep an eye on, but otherwise ignore.



Appendix A — Applying GlancePlus Data

This module is an optional self-study for students.

Objectives

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

- Use case studies to demonstrate how GlancePlus screens can be used to analyze system performance.
- Observe how a performance specialist approaches a tuning task.

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

The case studies stylized in this module come from the logbooks of HP-UX Performance Specialists and are presented for your consideration. The goal is to help you prepare for your own tasks and adventures. The examples show you possibilities and are not intended to be exact recommendations or solutions to situations that you may encounter. These examples may cause you to think up new questions, in addition to answering some of the classic tuning scenarios. As in most endeavors, there is often much to be gained from reviewing someone else's actions and trying to reverse-engineer their solutions.

An Approach to Monitoring System Behavior

The best approach to monitoring your system's performance is to become familiar with how your system *usually* behaves. This helps you recognize whether a sudden shift in activity is normal or a sign of a potential problem.

The first screen that appears when you start GlancePlus in character mode summarizes system-wide activity and lists all processes that exceed the usage thresholds set for the system. The information on this screen tells you if a resource is being used excessively or a process is monopolizing available resources. The Global screen is the usual starting point for any review of system activity and performance.

You can use the statistics on the Global screen to monitor system-wide activity, or you may need to refer to the detailed data screens to focus on specific areas of system usage. The examples in this chapter highlight the use of all GlancePlus screens.

GlancePlus provides you with valuable information, but optimal use of this information depends on how well you understand your system's operation and what is the normal or usual behavior for that system. As you use GlancePlus to review your system's performance, you will learn to recognize patterns that differ from this norm—patterns that may indicate a problem.

Bottlenecks

A bottleneck is the most common type of problem on any system. It occurs whenever a hardware or software resource cannot meet the demands placed on it, and processes must wait until the resource becomes available. This results in blocks and long queues.

Your system handles processes much like a freeway system handles traffic. During normal hours, the freeway adequately carries the traffic load, and cars can travel at optimum speed. But, during rush hour, when too many cars try to access the freeway, the lanes become clogged and traffic can slow to a halt. The freeway becomes bottlenecked.

Similarly, a bottleneck can occur on your system if the processes you are running need more CPU time than is available or more memory than is configured for the system. A bottleneck also can occur if there isn't enough disk I/O bandwidth to move data, or if swap space isn't configured optimally.

A bottleneck can be a temporary problem that is easily fixed. The solution may be to rearrange workloads, such as rescheduling batch programs to run late at night. Solving a disk bottleneck may require only spreading disk loads among all the available disks.

A recurring bottleneck, however, can indicate a long-term situation that is worsening. Perhaps the system was configured to serve fewer users than are now using it, or workloads have gradually increased beyond the system's capacity. The only solution may be a hardware upgrade, but how do you know?

If you can identify a bottleneck correctly, you can avoid randomly tuning the system (which can worsen the problem), and you can avoid adding extra hardware that doesn't help performance. You can also avoid expending resources solving a corollary bottleneck—one that is caused by the primary bottleneck.

Characteristics of Bottlenecks

Common system bottlenecks have several general characteristics or symptoms. By comparing these symptoms with the statistics on your GlancePlus screens, you can analyze the performance of your system and detect potential or existing bottlenecks. Although a single symptom may not indicate a problem, a combination of symptoms generally reflects a bottleneck situation.

Symptoms of a CPU Bottleneck

- Long run queue without available idle time
- High activity in user mode
- Reasonable activity in system mode (high activity may indicate other bottlenecks as well)
- Many processes frequently blocked on priority

Symptoms of a Memory Bottleneck

- High swapping activity
- High paging activity
- Very little free memory available
- High disk activity on swap devices
- High CPU usage in system mode

Symptoms of a Disk Bottleneck

- High disk activity.
- CPU is idle, waiting for I/O requests to complete
- High rate of physical reads/writes
- Long disk queues

Symptoms of Other I/O Bottlenecks

- High LAN activity
- Low I/O throughputs

You may discover that solving one bottleneck uncovers or creates another. It is possible to have more than one bottleneck on a system. In fact, changing workloads are constantly reflected in changing system performance. The goal is not to seek a final solution, but to seek optimal performance at any given time.

Evaluating System Activity

One afternoon Doug noticed that system response had slowed. He ran GlancePlus and looked at the **Global** screen to view system-wide activity. He saw that the CPU usage was near 100%. Although this is not necessarily a problem, he decided to check it out.

Doug then looked at the process summary section of the **Global** screen, which lists all processes that exceed the usage thresholds set for the system. He noted that a single process accounted for a majority of the near 100% CPU usage.

Wanting more information on that particular process, he checked it on the **Individual Process** screen, which provides detailed information about a specific process. Reviewing that screen, Doug noticed the process was doing no I/O and was spending all its time in user code. This suggested that the process might be trapped in a CPU loop. After identifying the user's name he telephoned the user to find out if the process could be killed.

In this situation, the CPU use for the system did not drop after the user terminated the looping process, because other processes took up the slack. However, response time improved because other processes did not need to wait as often to be given their share of CPU time.

Evaluating CPU Usage

Dean was checking the system one afternoon when he noticed a sudden slowdown in system response time. He ran GlancePlus and looked at the **Global** screen to view system-wide activity. He saw that the CPU usage was near 100%. The other system resources, such as Memory, Disk, and Swap, showed much less use. Further checking revealed that several processes were blocked due to another process using the CPU (PRI), which meant they were waiting for higher priority processes to finish executing.

Dean accessed the **CPU Detail** screen to see how CPU time was allocated to different states (activities). He discovered that real-time activities were using a much larger percentage of the CPU than other activities.

Dean returned to the **Global** screen to check priorities. One user was running with a priority of 127-- an RTPRIO real-time processing priority. Dean knew that this particular program is CPU-intensive and running it at such a high priority would keep other processes from executing. Already it was causing system performance to degrade. He reset the priority for that process to a lower timeshare priority by using the GlancePlus **renice** command. This allowed other processes more consistent access to the CPU, and system response time improved.

Evaluating Wait States

Jose's system was running fine until he installed a new application. Now, every time the application runs, response time degrades. Since the application is the only change to the system, Jose starts by checking how it is using the system. Looking at the **Glance Individual Process** screen, he sees that CPU utilization is about 7 percent, so that isn't the problem. Next, he checks overall CPU utilization on the system; it's averaging about 48 percent, which means there is sufficient CPU resource to accommodate the new application. Jose checks disk I/Os and notices the application is processing about 5 I/Os per second, most

of which is virtual memory I/O. That looks slow to Jose, so he looks at the Wait States screen to find out what the process is waiting on.

Jose learns that the process is spending about 7 percent of its time utilizing the CPU (executing), 27 percent of its time waiting for terminal input, and 66 percent of its time waiting on virtual memory. That's a significant amount of time. Jose checks other processes on the system and discovers that they are experiencing similar waits for virtual memory. He realizes that the new application overloads the system's memory. He makes copies of the relevant screens so he can explain the situation to his manager.

Evaluating Disk Usage

Vivian's company often runs processes that tax available memory. She keeps track of the situation by checking the **Disk Detail** screen, which displays both logical and physical I/O requests for all mounted disk devices. It also categorizes the physical requests as User, Virtual Memory, System, and Raw requests. This screen shows her when large numbers of physical read and write requests are occurring, a situation that results from excessive page faults by processes. Vivian also checks the virtual memory request rate, since that also will be high when system demand is taxing its physical memory capacity.

By paying attention to which processes are active when the virtual memory activity is high, Vivian can make intelligent decisions about redistributing activities to balance the system load. This helps increase overall throughput for the system.

Evaluating Memory Usage

Terri's system was experiencing a slowdown in system response time. She checked the **Global** screen to get an overall picture. All four system resources (CPU, Disk, Memory, and Swap) were near 100%. A large portion of the disk bar activity showed virtual memory activity. In addition, the swapper system daemon appeared to be running continuously.

Terri realized that this indicated a possible memory bottleneck. She checked the **Memory Detail** screen, which provides statistics on memory management events such as page faults, number of pages paged-in and paged-out, and the number of accesses to virtual memory. The screen indicated that Free Memory was 0.0 MB, indicating a lack of usable memory, and the Swap In/Outs showed a rate above 1 per second.

Concluding that the problem was a memory bottleneck, Terri returned to the **Global** screen to study the active processes. She knew that a memory bottleneck can be relieved by adding more memory or by reducing the memory demands of active processes. In this case, she suspected that high swap rates were caused by the large Resident Set Sizes (RSS) for the most active processes.

One test program showed a large RSS that appeared to grow at a constant rate. Examining this situation more closely, Terri discovered the program had a "memory leak." It allocated memory using `malloc()` but did not free up memory using `free()`. The process's memory allocation increased steadily, causing memory pressure on the system.

She talked with the developer, who studied the program code and found the memory leak. The test program was changed and recompiled to use far less memory, thus alleviating the memory bottleneck and improving system response time.

Evaluating I/O by File System

Ingrid noticed that system performance degraded drastically when the system was doing swapping. Looking at the **Global** screen, she observed that the swapper process was running and that virtual memory use counted for a high percentage of the disk utilization.

She checked the **Disk I/O by File System** screen to verify which disk was busiest. The **Disk I/O by File System** screen provides details of the I/O rates for each file system or mounted-disk partition. This information is useful for balancing disk loads. When she looked at the **Disk I/O by File System** screen, Ingrid saw that one disk was being utilized more than all other disks on the system. The disk most utilized was a swap disk.

Ingrid decided to add additional swap disk areas to the system to alleviate the load on that one disk. She also might have considered allocating dynamic swap areas on existing under-utilized file systems.

Evaluating Disk Queue Lengths

Ray had already determined that his system had a disk I/O bottleneck. By reading the **Global** screen, he noticed that the disk utilization was almost always at 100%. He had checked the **Disk I/O by File System** screen, which showed that several disks were being heavily utilized.

What Ray wanted to find was a way to ease the situation. He studied the **Disk Queue Lengths** screen, which shows how well disks are able to process I/O requests. He wanted to determine which of the busy disks had the longest delays for service. He knew that "busy" disks did not necessarily have a long queue length. High disk utilization is not a problem unless processes must wait to use the disk. For example, using a high percentage of the lines on a telephone system is not a problem unless calls cannot get through.

Ray also knew that long queue lengths meant several disk requests must wait while that drive is servicing other requests. For example, if all phone lines are busy, incoming calls must wait to connect.

Once he had a clear picture of the situation, Ray reduced the large queue lengths by moving several files to different file systems to distribute the workload more evenly.

Evaluating NFS Activity

Paul works on a system that is used as a network file system server. One local disk is NFS-mounted from several different nodes on the LAN. One afternoon, Paul noticed poor response time on the system. The file system mounted by remote systems was very active.

Paul reviewed the **NFS Detail** screen, which provides current statistics on in-bound and out-bound network file system (NFS) activity for a local system. He wanted to determine which remote system was using the disk the most. He observed a large In-bound Reads rate from one system. This led him to examine that remote system to find out why it was over-utilizing the NFS-mounted disk.

His examination pinpointed the situation to a single user on the remote system. The user was making repeated, unnecessary **greps** to files on the NFS-mounted disk. Paul explained the problem to the user and worked with her to lessen the heavy disk use. This reduced the load

on the NFS server and improved overall response time.

Evaluating LAN Activity

Lee noticed a slow response time for applications using datacom services to access data across the local area network. He checked the **LAN Detail** screen to see what was causing the problem. The **LAN Detail** screen describes four functions for each local area network card configured on the system. On networked systems, this information can show potential bottlenecks caused by heavy LAN activity.

Lee noticed that the **Collision** and **Errors** rates were higher than usual. This information led him to investigate whether processes were competing for LAN resources or overloading the LAN software or hardware.

In this case, an application that was improperly written using `netipc()` was causing a bottleneck. Once this program was stopped, other programs using the LAN were able to improve their response time.

Evaluating System Table Utilization

When Debbie was running a program on the system, the program failed, giving this error message:

```
sh: fork failed - too many processes
```

To decide whether or not to reconfigure the value of `nproc` in her kernel, Debbie needed to find out how much room she had in the Process Table. She referred to the **System Table Utilization Detail** screen, which provides information on the use and configured sizes of several important internal system tables. The information on this screen provides feedback on how well the kernel configuration is tuned to the average workload on the system.

Debbie confirmed that she had indeed run out of room in the Proc Table. She knew that usually the system buffer cache is fully utilized. Other tables can be proactively monitored in order to reconfigure the appropriate kernel variable before she reached a limit.

Evaluating an Individual Process

Cliff noticed that one process seemed to be running quite slowly. He ran GlancePlus and looked at the detail information for that process. He then examined the statistics on the **Single Process Detail** screen, which provides detailed information about a specific process.

Cliff knew that if the process was running slowly because of a memory shortage, he would see an increase in context switches and fault counts.

He noticed that the I/O read and write counts were large and that the process was doing a lot of I/O. He checked what the process had been blocking on and noticed a high percentage for Disk I/O blocks. He suspected that the process was slowed because of competition for disk throughput capacity.

Had the process shown a high percentage of being blocked on priority, it would have meant the process was ready to run but was unable to do so because the CPU was being used by processes with higher dispatching priorities.

Evaluating Open Files

Kathryn is developing an application for communicating with remote systems. When a request is received, the application opens a socket and sends the specified data. However, when Kathryn tests the application, no data is received by the remote system. To find out what happened she checks the **Glance Open Files** screen. When she looks for the opened socket, she discovers that it never opened. She returns to her application to look for the coding error.

Evaluating Memory Regions

One day while reviewing Glance's **Global Summary** screen, Nancy notices that several processes have very large resident set sizes. Could this mean a potential problem with the applications' memory usage? She wonders if she should begin planning to increase physical memory size to accommodate additional users in the future. She knows current system performance is fine and memory size seems adequate, but she wants to prevent any future degradation in performance.

Before making any decisions she reviews Glance's **Memory Regions** screen to analyze the situation more closely. She discovers that all of the affected processes have a shared memory region of about 200 KB. When added to the private DATA and TEXT regions, this accounts for the large resident set sizes. By checking the virtual address of the **shmem** region, she determines that the same shared memory region is being used by the processes. Because it is a shared region, it is physically in memory only once, but Glance displays it for each process attached to the shared memory region. Nancy smiles when she sees this, because it means that no problem exists. By using the shared memory region the processes are using far less memory than it appears.

Evaluating All CPUs Statistics

Rosalie works on a multiprocessor system. While checking the **All CPUs** screen one day she noticed that one CPU seemed to be consistently busier than the others. Realizing that overall system throughput would be improved if the load were balanced among the processors, she decided to investigate the situation. As she studied the **All CPUs** screen she noticed that one PID always seemed to be the last PID executing on CPU 1, her busiest CPU. When Rosalie checked **mpctl()**, she saw that the process had been assigned to CPU 1. Using **mpctl -f**, she reassigned the process to be a floater, so that the system could determine which processor should run the process. Rosalie then checked for other processes that had been assigned to CPU 1 and reassigned them as floating processes. After doing so, she rechecked the **All CPUs** screen and observed that the load appeared more even among all the processors, thus alleviating a potential bottleneck on any single CPU.

Evaluating Activity on Logical Volumes

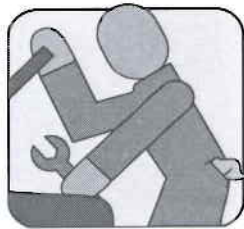
Lately when Yuki uses GlancePlus to check his system, he notices that the **Global Disk Utilization** bar — displayed in the top portion of every Glance screen — is often close to 100%. Yuki's system has multiple disk drives, but he knows that the global disk utilization figure indicates the activity on the busiest disk.

Yuki would like to spread the disk I/O more evenly among the drives to avoid potential I/O bottlenecks. With that goal in mind, he first checks the **Disk Detail** screen. It shows that logical disk activity is high. For details, he goes to the **Logical Volumes** screen, where he notices a high write activity on logical volume `/dev/vg00/lvol12`. Getting out of Glance and into the UNIX shell, he types `vgdisplay -v /dev/vg00` to ascertain the physical disk names associated with the volume.

Back in Glance, Yuki views the **Disk Queue Lengths** screen to determine the busiest disks in the volume. Then he checks the **Disk Detail** screen to find out whether disk activity was caused by system or user activity. Yuki notices that the Virtual Memory physical accesses are low, indicating application rather than system activity. He checks the **Open Files** screen to find which application was creating so many writes to the disk. Voila! Fred is running his baseball pool again! Yuki pays a visit to Fred.

After discussing Fred's I/O needs, Yuki returns to his console to balance the I/O load, using LVM commands to rearrange the logical volumes.

Now it's time to grab your toolbox, pop the hood, and take a look.



Good Luck!

Appendix A Applying GlancePlus Data



Solutions

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:
```

Answer:

Varies with system configuration – in the order of 10's of seconds to minutes.

Example output follows from an rp2430 server:

```
# timex ./long
The last prime number is : 49999
```

```
real      3:37.89
user      3:35.68
sys       0.12
```

Example output follows from an rx2600 server:

```
# timex ./long
The last prime number is : 99991
```

```
real      2:53.24
user      2:51.74
sys       0.06
```

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

```
# timex ./med
```

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

Answer:

Varies with system configuration – should be about one half of **long**.

Example output follows from an rp2430 server:

```
# timex ./med
The last prime number is : 49999

real      1:52.68
user      1:51.55
sys       0.08
```

Example output follows from an rx2600 server:

```
# timex ./med
The last prime number is : 99991

real      1:33.71
user      1:33.02
sys       0.04
```

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:
```

Answer:

Varies with system configuration – should be about one eighth to one tenth of **med**.

Example output follows from an rp2430 server:

```
# timex ./short
The last prime number is : 49999

real      10.88
user      10.70
sys       0.05
```

Example output follows from an rx2600 server:

Solutions

```
# timex ./short
The last prime number is : 99991

real      8.56
user      8.49
sys       0.03
```

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

```
# timex ./diskread

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

Answer:

Varies with system configuration – in the order of tens of seconds.

Example output follows from an rp2430 server:

```
# timex ./diskread
DiskRead: System      : [HP-UX]
DiskRead: RawDisk     : [/dev/rdisk/clt15d0]
DiskRead: Start reading : 1024MB
1024+0 records in
1024+0 records out

real      28.01
user      0.02
sys       0.53
```

Example output follows from an rx2600 server:

```
# timex ./diskread
DiskRead: System      : [HP-UX]
DiskRead: RawDisk     : [/dev/rdisk/c2t1d0s2]
DiskRead: Start reading : 2048MB
2048+0 records in
2048+0 records out

real      28.69
user      0.01
sys       0.13
```

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.

Answer:

We first assume that there is no other load on the system. In the case of a classic number crunching CPU hog (**long**, **med**, and **short** are all these) there will be no system calls (except for the final terminal output) and the program only needs CPU time in **usr** mode.

As there is only one process, there is no waiting. This is shown by the “real” time being very close to the sum of the **sys** and **usr** times for the process.

long, **med**, and **short** only do calculations and make no call on kernel resources during their execution, so the **usr** time is very high compared to the **sys** time.

This is not the case for **diskread**. The program makes very little demand on the CPU, shown by the sum of **usr** and **sys** being quite small compared to the real or “wall clock” time. The huge difference between real time and **usr+sys** time proves that the program is waiting on disk I/O most of the time. Also note that **sys** is much higher than **usr** meaning that the program is bound on system calls (disk I/O) rather than computation when it does execute.

1-12. LAB: Verifying the 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`).

Example figures below are from a C200 workstation

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 &
```

Answer:

rp2430:

```
# timex ./short &
[1]      10050
```

```
# The last prime number is : 49999
```

```
real      10.85
user      10.70
sys        0.05
```

```
# timex ./short &
[1]      6486
```

rx2600:

```
root@r265c145:/home/h4262/baseline
# The last prime number is : 99991
```

```
real      8.59
user      8.50
sys        0.03
```

How long did the program take to execute?
8 to 11 secs.

How does this compare to the baseline measurement from earlier?
A little longer due to the overhead of **sar**.

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? _____

How did the CPU queue size change from first window? _____

Answer:

rp2430:

```
# timex ./short & timex ./short & timex ./short &
[1]      10203
[2]      10205
[3]      10206
```

```
# The last prime number is : 49999
```

```
real      29.86
user      10.68
sys       0.01
```

```
The last prime number is : 49999
```

```
real      32.07
user      10.67
sys       0.01
```

```
The last prime number is : 49999
```

```
real      32.35
user      10.67
sys       0.01
```

rx2600:

```
# timex ./short & timex ./short & timex ./short &
[1]      6690
[2]      6692
[3]      6694
```

```
# The last prime number is : 99991
```

```
real      25.08
user      8.48
sys       0.00
```

```
The last prime number is : 99991
```

```
real      25.56
user      8.48
sys       0.00
```


Solutions

The last prime number is : 99991

```
real      25.60
user      8.48
sys       0.00
```

How long did the slowest program take to execute?

25 to 34 secs, around three times longer than one occurrence of the program.

If you have a multiprocessor, the time will be distributed over the number of processors – with the lower limit being the time a single process would take. For example, if your system had two processors, the slowest process would complete in one-half the time it would take on a single-processor system. Since we're only running three processes here (not including **sar**), three processors or more than three processors would show the same results.

How did the CPU queue size change from first window?

The **sar -q** shows that the average cpu queue length (first field) increases by three times when three programs are run concurrently.

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? _____

How did the CPU queue size change from first window? _____

Answer:

rp2430:

```
# timex ./short & timex ./short & timex ./short & \
  timex ./short & timex ./short &
[1]      10212
[2]      10214
[3]      10216
[4]      10218
[5]      10220
```

The last prime number is : 49999

```
real      53.98
user      10.68
sys       0.01
```

The last prime number is : 49999

```
real      54.08
user      10.68
sys       0.01
```

Solutions

The last prime number is : 49999

```
real      54.08
user      10.68
sys       0.01
```

The last prime number is : 49999

```
real      54.08
user      10.67
sys       0.01
```

The last prime number is : 49999

```
real      54.15
user      10.68
sys       0.01
```

rx2600:

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

```
[1]      6737
[2]      6739
[3]      6741
[4]      6743
[5]      6745
```

The last prime number is : 99991

```
real      42.52
user       8.49
sys       0.00
```

The last prime number is : 99991

```
real      42.56
user       8.48
sys       0.00
```

The last prime number is : 99991

```
real      42.59
user       8.48
sys       0.00
```

The last prime number is : 99991

```
real      42.67
user       8.48
```

Solutions

```
sys          0.00
```

```
The last prime number is : 99991
```

```
real        42.75
user         8.48
sys          0.00
```

How long did the slowest program take to execute?
43 to 54 secs

If you have a multiprocessor, the time will be distributed over the number of processors – with the lower limit being the time a single process would take. For example, if your system had two processors, the slowest process would complete in one-half the time it would take on a single-processor system. Since we're only running five processes here (not including `sar`), five processors or more than five processors would show the same results.

How did the CPU queue size change from first window?
It increased by 5 while the test is being run.

5. Is the relationship between elapsed execution (real) time and the number of running programs linear?

Answer:

Yes very much so. The fastest program in the last case (where 5 programs are running) takes five times longer than with one program. You can draw a graph and go to 10 programs if you are unsure! Typing the command with more than 10 occurrences gets a little tedious! You will find a linear relationship in any case.

6. Comment about the overhead of switching from one process to another.

Answer:

The overhead of task switching is very low. If it were not, the relationship in the above tests would not be linear. If there is an overhead, it looks like we will not see it unless there are hundreds of processes being switched.

2-68. LAB: Performance Tools Lab

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.

These results were obtained on a C200 workstation running 11i. Remember the absolute numbers are not important here but you should be drawing similar conclusions.

1. How many processes are running on the system?
Which tools can you use to determine this?

Answer:

top	Gives the number of running processes in the summary portion of the screen: <i>119 processes: 96 sleeping, 17 running, 6 zombies</i>
ps	<code>ps -e wc -l</code> and subtract 1 for the headers and 1 each for ps and wc .
glance	Look at the table screen (t page) and see the current size of the proc table.
sar	<code>sar -v 2 10</code> Look at the proc-sz field.
gpm	Gives the count at the top of the Process List report.

2. Are there any real-time priority processes running? If so, list the name and priority. What tools can you use to determine this?

Answer:

syncer, **midaemon**, **lab_proc2**, sometimes **swapper**.
ttisr and **prm3d** will also be seen on 11/11i systems running at **pri -32**. This is the posix real time range which is even higher than the normal UNIX real time priorities.

glance	Global, PRI column (Turn off all filters)
top	PRI column

Solutions

gpm Use the filters to filter priorities <128 (process list/configure/filters)
ps -el PRI column

(Try this command:

```
# ps -el | grep -v PRI | sort -k 7,7n | more
```

The highest priority processes will be listed on the top.)

Remember a real time priority is anything less than 128

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?

Answer:

glance Go through each single process screen.
(Default is 20. 21- 39 is nice; 0 to 19 is nasty, i.e. anti-nice.)
gpm Process list, select a process by double clicking.
top NI column
ps -el NI column

(Try this command:

```
# ps -el | grep -v NI | sort -k 8,8n | more
```

The nasty processes will be listed at the top and the nice'd processes will be at the bottom.)

On **11i** the following were "nasty":

diagmond, diaglogd, psmctd, memlogd, krsd

The following were "nice":

All 6 <defunct> zombie processes (see below), **lab_proc4**

4. Are there any zombie processes on the system? If so, how many are there? What tools can you use to determine this?

Answer:



A zombie is a terminated process whose parent is still running, but has not called **wait()** for the child.

Zombies whose parent has terminated should eventually be adopted by the **init** process, which will issue a **wait()** on the zombie. Therefore, a zombie whose parent has terminated should eventually disappear.

What resources do zombies consume? **memory (<= 20 pages), table entries**

<code>top</code>	The number of zombie processes is shown in the summary portion of the screen: <i>119 processes: 96 sleeping, 17 running, 6 zombies</i>
<code>glance</code> and <code>gpm</code>	By design, they do not currently report zombies, unless the process entered the zombie state during the interval.
<code>ps -el</code>	Z in S(tate) column and <defunct> in the Comm(and) column

5. What is the length of the run queue? What are the load averages? What tools can you use to determine this?

Answer:

<code>glance</code>	CPU screen (c), page 2, shows a RUNNING LOAD AVERAGE, but has been labeled incorrectly in older versions as run-queue. The All CPUs screen (a) shows the 1-, 5-, and 15-minute load averages whereas the CPU screen shows the interval load average.
<code>gpm</code>	CPU button or Reports/CPU Graph show the interval load average. Reports/CPU report shows the interval load average. Reports/CPU by processor shows the 1-, 5-, and 15-minute load averages.
<code>uptime</code>	1-, 5-, and 15-minute load averages
<code>top</code>	Load averages: 5.39, 5.27, 5.20 and interval load average
<code>sar -q</code>	Average run queue size over interval.
<code>vmstat</code>	r column is the run queue size over the interval.
<code>xload</code>	10-second load average over time.

The run queue length on the test system was around 5 no matter how it was measured.

There is also a hardware dependant approach that can be used on servers using the console Hex display code....



HEX display (front panel or console) shows size of `runQ` in the second digit. F31F means there are three processes in the `runQ` and one CPU. FA1F means 10 or more in the `runQ`. MPE uses this as a percent utilization number.

The `runQ` is an instantaneous value and can never be a fractional number. The load average is based on the `runQ`, but includes short sleepers (discussed in CPU section).

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. (i.e. `swapper`, `vhand`, `statdaemon`, `unhashdaemon`, `supsched`, etc.) `ps` reports their size as zero. Others as below.

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.

Solutions

Answer:

top	PA-RISC: RES = 16K (32-bit kernel) or 32K (64-bit kernel) per thread. IA-64: RES = 80K per thread
glance	PA-RISC: 16K (32-bit kernel) or 32K (64-bit kernel) per thread on global screen. IA-64: 80K per thread on global screen.
ps -el	The second bit in the F column value indicates a system process. (See the man page for ps .) F column = 3, PPID column = 0, and SZ column = 0.

(Try this command: # **ps -el | grep " 3 " | more**
This will list all the system processes. No, technically, **init** is NOT a system process.)

This amounts to 17 processes on the test **11i** system.

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

Answer:

glance	Bar graph, CPU screen (c): displays detailed CPU state information. Per-process (s): details per/process CPU utilization. (Display can be toggled between cumulative/interval (C) and percent/absolute (%).)
gpm	Main window → Reports/CPU or Reports/CPU by processor
sar	user/system/waiting for io/idle
top	user/nice/system/idle/block/swait/intr/ssys (context switch) SEE /usr/include/sys/dk.h for CPUSTATE (CP_USER, etc.). block is spinlock percentage (on MP systems only). This figure is obsolete at 11i . swait is alpha semaphore percentage (on MP systems only). This figure is obsolete at 11i .
vmstat	user/system/idle
iostat -t	us/ni/sy/id

NOTE: Always watch the first line in **vmstat** or **iostat**. It is the average since bootup. Use **vmstat -z** to clear the sum structures for **vmstat**. There is no similar option for **iostat**.

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

Answer:

glance	M(emory) screen, Free Memory, Phys Memory, Avail Memory, Total VM, Active VM, Buf Cache Size
gpm	Reports/Memory Report
vmstat	Free (in pages) avm (active virtual memory, in pages) includes on-disk pages

top Real (real active), virtual (virtual active), free in KB.
/etc/dmesg Amount of physical and available memory.

The memory stats from **top** are misleading. The values in brackets are figures for processes that are regarded as “busy” whatever **top** means by that. In most utilities, “busy” or “active” means that the process is in the RUN state, or has executed within the last 20 seconds. The “real” figures are a summation of the resident set sizes for all processes (sum of the RES field). This is not the amount of physical memory in the system.

The only way to get the true physical memory in the system is through **glance/gpm** or **dmesg**. The boot info seen in **dmesg** with the physical memory figure will be lost if general console messages (e.g. file system full) have overwritten the limited buffer space.

vmstat figures are generally accurate with the “free” field agreeing well with **glance/gpm** and **top**. Remember **top** reports memory in 1K units while **vmstat** reports memory in 4K pages so multiply the **vmstat** figures by 4 to compare them to **top**.

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?

Answer:

glance Bar graph (reserved/used) w(swap):%used (device and filesystem) and MB reserved by swap device, MB available and MB used
gpm Reports/Swap space (**glance w**) Reports/System Table Info/System Tables Graph Report

NOTE: Graph shows high water mark (nice!)

sar -w N/A (only shows size of swap queue and swapping rates)
vmstat -S N/A (only shows paging and swapping rates)
top: N/A
swapinfo -t KB avail/used/free/% used by swap device
bdf -b File system swap space used/avail

swapinfo can be misleading unless you know what you are looking at. To remove the confusing issue (pseudo swap) enter a **swapinfo -t** to correctly calculate and include pseudo swap issues by taking the “total” figures. This will be explained in detail in the module on swap space management.

```
# swapinfo -t
      Kb      Kb      Kb      PCT  START/      Kb
TYPE   AVAIL   USED   FREE  USED  LIMIT RESERVE  PRI  NAME
dev    524288   11276  513012    2%      0      -    1  /dev/vg00/lvol2
reserve -   204360 -204360
memory 180940   84488   96452   47%
total  705228  300124  405104   43%      -      0    -
#
```

Note that we have used 43% of our swap space and not 2%.

Solutions

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

Answer:

glance	t(able) screen, page two
gpm	Reports/System Table Info/System Tables Report (NOT Graph)
sar -v	Used/size/overflows



Remember that the inode cache may contain entries for files that are closed, if it doesn't need to flush it out to open a new file

Its size is the maximum number of unique files that can be open system wide.

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?

Answer:

glance	Global screen and single process screen
gpm	Can sort by CPU utilization and can filter by > 0
top	Automatically lists the processes by CPU utilization
ps -el	Cpu hogs often have large C counts (<=255)

(Try this command:

```
# ps -el | grep -v " C " | sort -k 6,6n
```

This will list the most active processes at the end.)

lab_proc5 and **lab_proc3** are the main CPU users. They are consuming close to 100% of the CPU between them. This is not normal behavior!

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?

Answer:

glance	Global screen: RSS, Per-process screen: RSS and VSS sizes
gpm	Reports/Process List (glance g)
top	SIZE (KB: text/data/stack), RES (KB: resident size)
ps -el	SZ column (size in 1-K blocks of the core image including only text + data + stack)

(Try this command:

```
# ps -el | grep -v SZ | sort -k 10,10n
```

This will list the largest processes at the end.)

lab_proc1 has a much larger SZ (**ps -el** output) size than most other processes. This program is 8MB in core and could be regarded as a memory hog. Remember that SZ is in pages, multiply by 4K to get the actual figure.

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.

Answer:

glance i screen will periodically show **lab_proc3** as largest disk user
s(ingle) process screen, open files, will show actual open files and offset, which MIGHT be indicative of the amount of I/O

gpm Reports/Process List

sar -d Reports physical disk I/O for the system overall.

sar -b Reports and compares logical I/O to physical I/O

Notice **sar -b** reporting very high logical read I/Os. The **lab_proc3** process is very busy with disk reads but the system has cached all the data in the buffer cache preventing physical disk I/O.

```
# sar -b 2 2
```

```
HP-UX workstn B.11.11 U 9000/782 01/22/01
```

```
15:39:38 bread/s lread/s %rcache bwrit/s lwrit/s %wcache pread/s pwrit/s
15:39:40      0    19646     100      1      2      60      0      0
15:39:42      0    21454     100      0      3      83      0      0
```

14. What is the current rate of semaphore or message queue usage? What tools can you use to determine this?

Answer:

sar -m The ONLY tool to show message and semaphore ops/sec.

glance The single process screen shows messages sent/received.

Semaphore and message usage was effectively zero in the lab test as none of the test programs manipulate semaphores or messages. These resources will be covered in a later module. Relational data bases (Oracle, Informix, Sybase etc) are big users of such resources.

```
# sar -m 2 2
```

Solutions

HP-UX workstn B.11.11 U 9000/782 01/22/01

```
15:41:58    msg/s    sema/s
15:42:00      0.00      3.98
15:42:02      0.00      2.00
```

15. Is there any paging or swapping occurring? What tools can you use to determine this?

Answer:

glance	m(memory) screen: page faults, paging request, KB paged in, KB paged out, deactivations/reactivations, KB swapped in, KB swapped out, VM reads, VM writes
gpm	Reports/Memory Graph (or Memory button): page OUTS, swap OUTS Reports/Memory Report: in/out/etc.
sar -w	Swapping only
vmstat	Paging (pi/po)
vmstat -S	Swapping (si/so) & Paging (pi/po)

In terms of simple UNIX commands, **vmstat** is the way to go. The **sar** command does not understand paging (more in the module on memory management!) and is measuring the swap rate only. See the **pi** and **po** fields below from **vmstat**. This system is not paging at all so we can be confident that there is no swapping activity.

```
# vmstat 2 3
      procs
      faults      cpu
      r    b    w    avm    free    re    at    pi    po    fr    de    sr    in
      sy    cs    us sy id
      4      0      0    78478    1593      6      1      0      0      0      0      0    108
2096    171      3      2  94
      4      0      0    78478    1552      2      0      0      0      0      0      0    117
```

glance/gpm do give good paging and swapping detail on the "m" screen and the data should tie in with **vmstat**.

16. What is the system call rate? What tools can you use to determine this?

Answer:

glance:	CPU screen, page 2; Single process screen (s), then (L) reads/writes/opens/closes/ioctls/forks/vforks/messages sent and received
gpm	Reports/CPU Report Reports/Process List, select for single process screen
sar -c	Total/reads/writes/forks/execs
vmstat	First sy column (under "faults")

sar and **vmstat** give good data that should agree here. The system call rate can be used as an indication of how busy your system is once you have established the normal range for its value. There is no absolute "good" or "bad" figure as this depends on:

- a) How powerful your system is.
- b) How many cpus you have.
- c) What processes you are running.

Example live data from the test system (C200 running 11i):

```
# sar -c 2 10
```

```
HP-UX workstn B.11.11 U 9000/782 01/22/01
```

	scall/s	sread/s	swrit/s	fork/s	exec/s	rchar/s	wchar/s
16:03:24							
16:03:26	4417	1205	1188	0.00	0.00	85899336	2038
16:03:28	4623	1252	1249	0.00	0.00	88524288	4096

Note the system is doing over 4k system calls per second (scall/s), over half of which can be attributed to reads (sread/s) and writes (swrit/s).

See how dramatically this number can be changed by adding a simple extra process (you might like to try this while monitoring **sar -c**).

```
# dd if=/stand/vmunix of=/dev/null bs=64 &
```

```
# sar -c 2 10
```

```
HP-UX workstn B.11.11 U 9000/782 01/22/01
```

	scall/s	sread/s	swrit/s	fork/s	exec/s	rchar/s	wchar/s
16:10:37							
16:10:39	21528	10461	9586	4.98	4.98	172712128	16302
16:10:41	19882	7790	6878	5.00	5.00	148155392	7168

The **glance/gpm** tools become invaluable when you want to know which processes are hitting the system with so many system calls. More on this later.

17. What is the buffer cache hit ratio? What tools can you use to determine this?

Answer:

sar -b	Read and write hit ratios
glance	Disk screen (d), page 2 shows both ratios
gpm	Reports/Disk report shows both ratios

See answer 13 for example output of **sar -b**.

18. What is the tty I/O rate? What tools can you use to determine this?

Answer:

```
sar -y:
iostat -t:
```

The quickest tool to use here is **iostat -t**. In general modern system administrators care less and less about terminal I/O as almost all users connect to application servers and services over LAN networks. An exception to this rule would be the case of modems. A

Solutions

system with multiple modems may experience a modem “storm” with meaningless data being fired at the host by a bad modem line.

`iostat -t` will catch this problem as a high `tin` (tty characters read per second) value.

```
# iostat -t
          tty              cpu
      tin tout          us  ni  sy  id
          0    5          3   1   3  94
```

19. Are there any traps (interrupts) occurring? What tools can you use to determine this?

Answer:

`vmstat -s` Traps since bootup (should probably zero it out first)

NOTE:	examples of trap call
	page faults
	overflow/underflow (integer and floating point)
	HPMC/LPMC
	floating point emulation traps

When traps occur, the normal flow of a program is interrupted and work is done to take care of a problem before normal program instructions can be continued. For example, trying to access a data page which is not in memory and is out on disk would result in a “page fault” causing the execution of the program to stop, waiting for the required data to come in from disk.

Like with system call rates (see 16), there is no “good figure” but you are advised to monitor the trap rate as a “sanity” reference.

Clear the `vmstat` counters with `vmstat -z` first. Note that the numbers seen have been generated in the time between the two `vmstat` commands!

Example data from our C200 running 11i. The parameter list has been reduced and the trap events are in bold.

```
# vmstat -z
# vmstat -s
12 swap ins
12 swap outs
0 pages swapped in
0 pages swapped out
8636 total address trans. faults taken
2633 page ins
0 page outs
20 pages paged in
0 pages paged out
6594 cpu context switches
7640 device interrupts
11335 traps
153724 system calls
```

20. What information can you collect about network traffic? What tools can you use to determine this?

Answer:

glance	1(an) screen: packets in/out, collisions, errors NFS global screen (N): rd/wr rates, calls, response time, etc. nfs by system (n): reads/writes/response time by system for both client and server requests
gpm	Reports/LAN Graph (or NW button): packets in/out per second, Reports/Network by LAN, Reports/NFS Global Activity, Reports/NFS by system, Reports/NFS by operation
netstat:	Sockets in use -m memory buffers in use (NOTE: No longer works in HP-UX 11.X) -i packets in/out, errors in/out, collisions by interface -s packets, bytes, retransmissions, duplicate, acks, checksum errors, timeouts, etc. -rs routine statistics
nfsstat	Server rpc and nfs stats; client rpc and nfs stats

A later module will cover networking performance issues in more detail. The most important performance metric in a CSMA/CD (Ethernet or 802.3) network is collision rate. This is available in **glance/gpm** and in the Network module we will learn how to extract this data using **lanadmin**.

21. What information can be gathered on CPUs in an SMP environment?
What tools can you use to determine this?

Answer:

glance	a(11)CPU detail utilization and load averages by CPU
gpm:	Reports/CPU Info/CPU by Processor: util , ld avg , CS rate, fork rate, last PID
sar -Mu/Mq	Utilization/queue lengths by CPU (with -u or -q)
top	CPU number on which a process is assigned and utilization per CPU.

sar -M output has changed at 11i. The output of **sar -M** will look identical to **sar -u** (or **-q**) if the system only has one cpu. For MP systems you are presented with the **sar (-u or -q)** data on a per CPU basis. This becomes helpful in measuring the balance of processes across processors.

top displays its MP information by default, giving the cpu reference for each process. This information is hidden if there is only one processor or if the **-h** option is used.

The **a** page of **glance** also measures the balance per CPU and indicates the last process to run on any given CPU. Very useful.

Solutions

22. What information can be gathered on Logical Volumes?
What tools can you use to determine this?

Answer:

glance	v(LVM) screen reads/writes/MWC hits and misses by LV or VG
gpm	Reports/Disk Info/I/O by LV
vgdisplay, lvdisplay, pvddisplay	General information on Volume Groups, Logical Volumes and Physical Volumes. Use -v for details
bdf, mount	Information on file systems on logical volumes.

Physical disk layout (the positioning of data on disk) is important for performance. The **lvdisplay -v** and **pvddisplay -v** commands are the best way of finding out the precise layout of logical volumes on physical disks. In a later module we will look in detail at mirroring and striping techniques used to manipulate physical disk layout to our advantage.

23. What information can be gathered on Disk I/O? What tools can you use to determine this?

Answer:

glance	d(isk)	logical/physical reads/writes, user/VM/system/raw/NFS
	i(o):	by file system, logical/phys/VM
	u(queue)	queue length and utilization by spindle
	v(LV):	see above
gpm	press disk bottleneck button (queue)	
	Reports/Disk Info/Disk Report	(glance d)
	I/O by disk	(glance u + type [phys, logl, VM, FS, System, RAW])
	I/O by fs	(glance i + blocksize, util, logl, sys, VM)
	I/O by LV	(glance v)
iostat	KB/sec, seeks/sec, millisec/seek by spindle	(NOTE: millisec/seek no longer reported – permanently reports 1.0)
sar -d	%busy, average queue, io per sec, blocks/sec, average wait time, average service time	

iostat is a redundant tool because its data is not as useful or as accurate as that obtained from **sar -d**. The most important place to start looking for disk I/O info lies with the disks themselves. **sar** cannot understand LVM layouts and only “sees” the disk as a whole. Use **glance/gpm** on the individual disks once you have identified them with **sar -d**.

Below is some example data collected at the start of the tools lab. Stop the lab with **./KILLIT** and start it again with **./RUN** to see some disk I/O.

```
# ./KILLIT
Killing the lab procs
Removing the files
# ./RUN
cc -wall +DAportable cpu_hog.c -o cpu_hog
cc -wall +DAportable vm_bnd.c -o vm_bnd
cc -wall +DAportable io_bnd.c -o io_bnd
```


Solutions

```
cc -wall +DAportable zombie.c -o zombie
# sar -d 2 4
```

```
HP-UX workstn B.11.11 U 9000/782 01/22/01
```

	device	%busy	avque	r+w/s	blks/s	avwait	avserv
17:22:54	c0t6d0	3.50	0.50	6	160	3.55	9.24
17:22:58	c0t6d0	2.50	0.50	4	84	1.90	10.75
17:23:00	c0t6d0	3.00	0.50	5	84	2.57	8.75
17:23:02	c0t6d0	5.50	0.50	6	159	3.76	14.07
Average	c0t6d0	3.62	0.50	5	122	3.04	10.90

We would not consider 3-5% busy as being a bottleneck here. We will see much higher disk loads later!

Shut down the simulation by entering:

```
# ./KILLIT
```

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.

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

Solutions

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 ON

Change Filter Relation to >=
 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

Solutions

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.

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	

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 &
[1]      15722
[2]      15723
[3]      15724
[4]      15725
[5]      15726
[6]      15727
[7]      15728
```

3. Start a **glance** session. Answer the following questions.

How much CPU time is each **long** process receiving? _____sec _____%

Answer:

Hint: Change the sample period to 10 secs (hit the j key). This will give you more time to think and makes "per second" calculations easier!

The CPU should be balanced between the seven processes with each getting around 14% of the CPU (i.e. 5/7 seconds each for a 5 second interval and 10/7 seconds each for a 10 second interval). This is seen in the CPU Util field of the main **glance** window. Notice that the programs all have similar priority around 248-249 which is towards the bottom of the pile.

If you have a multiprocessor, the processes will quickly distribute themselves among all available processors. However, the overall metrics should stay the same – with the exception of the overall length of time that the processes take.

How are the processes being context switched (forced or voluntary)? _____

Answer:

Select one of the “long” processes using the **glance s** key. Make sure the PID being suggested is the right one or enter the correct PID.

In the first column of info you will find the “Forced CSwitch” and “Voluntary CSwitch” metrics.

You will notice that (almost!) all context switched are forced when you compare the two figures. This is normal for a CPU hog process. It never leaves the CPU on his own accord and is always told to leave by the scheduler.

We saw 7.7-9.6 context switches per second for the period for each of the processes on an rp2430. All of the context switches were forced.

On a multiprocessor, there would be the same number of context switches taking place, however fewer processes would be sharing the same processor.

How many times over the interval is the process being dispatched? _____

Answer:

Again, we can look to the first column of the selected process’ resource summary page. Find the “Dispatches” metric. This is a measure of how often the process is getting onto the CPU with the summation of “Forced CSwitch + Voluntary CSwitch” measuring how often the process gets switched out.

On a multiprocessor, each processor would have fewer processes wanting its resource, so, each process would be selected more often.

What is the ratio of system CPU time to user CPU time? _____

Answer:

Look to the first column of the selected process info again and you will find the “System CPU” metric. This will be zero or close to zero on any system. By using the C (upper case) key we can switch between metrics for the last interval (10 seconds if you are following the solutions) or the total over the period of tracking. It makes no difference how you look at it, these processes do not process system calls. They are typical CPU hogs that crunch numbers and do nothing else. All the CPU is User/Nice/RT.

What are the processes being blocked on? _____

Answer **PRI**ority

The most frequent event that is blocking the process is shown by the “Wait Reason” metric at the bottom of the first column of Process Resource info (the same page we have been looking at all along). In this case it is PRI, short for Priority.

Solutions

The process has been blocked because it is timeslicing with all the other processes. Each time it is switched out, it is placed at the end of the queue in true round-robin fashion. Thus, it is no longer the most eligible process to run and the scheduler has chosen another.

For more stats go to the Wait States page for this process (softkey **F2** or hit **W**) notice that the process is blocked on Priority for 80-90% (6/7) of the time and the rest of the time it is on the CPU. There are no other active wait states.

The seven long processes are in a circular fight to get to the top of the pile(s).

What are the nice values for the processes? _____

Answer 24

A Bourne-based shell (Bourne, Korn, Posix, bash) always places background processes at a nice level 4 higher than the calling shell. The standard nice value of our shell is 20 so the child background jobs inherit 24 as the nice value. One exception is the C shell which runs background processes at the same nice value as the shell.

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? _____

Answer:

To change the processes nice value, enter:

```
# renice -n -5 <PID of selected process>
```

Be careful! This forces a negative offset of 5 from 20 (the standard nice value) and not the current nice value (24). The nice value in this case will end up at 15, which is more favorable than the others, still at 24.

Watch that process's percentage of the CPU over several display intervals with glance or top. What effect did it have on the process? _____

Answer:

The effect on the process is that it will race away from the others, consuming approx 50-60% of the CPU! This might take a little time to settle down at 50-60%. Give it several intervals to complete its adjustment.

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? _____

Answer:

This really turns the process into a loser! The priority of the process drops to 251-252, preventing the process from getting much action. If you select the process and look in the first column of the Process Resource page you will see that it is being dispatched but not very often. You will see the process getting less than 2% of CPU but not much more. Each of the other processes will take up the excess, with the majority of the excess going to the process with the nice value of 15.

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)
```

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)
# ioscand -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 _____ user _____ system _____
```



```
# timex dd if=/stand/vmunix of=/dev/null bs=2k
real _____ user _____ system _____

# timex dd if=/stand/vmunix of=/dev/null bs=64k
real _____ user _____ system _____
```

Answer:**Results for an rp2430:**

```
# timex dd if=/stand/vmunix of=/dev/null bs=64k
282+1 records in
282+1 records out
```

```
real      0.04
user      0.00
sys       0.03
```

```
# timex dd if=/stand/vmunix of=/dev/null bs=2k
9055+1 records in
9055+1 records out
```

```
real      0.15
user      0.02
sys       0.12
```

```
# timex dd if=/stand/vmunix of=/dev/null bs=64k
289765+1 records in
289765+1 records out
```

```
real      3.82
user      0.56
sys       2.95
```

Results for an rx2600:

```
# timex dd if=/stand/vmunix of=/dev/null bs=64k
728+1 records in
728+1 records out
```

```
real      0.03
user      0.00
sys       0.03
```

```
# timex dd if=/stand/vmunix of=/dev/null bs=2k
23299+1 records in
23299+1 records out
```

```
real      0.18
user      0.02
```

Solutions

```
sys          0.13
```

```
# timex dd if=/stand/vmunix of=/dev/null bs=64
```

```
745575+1 records in
745575+1 records out
```

```
real        4.57
user        0.54
sys         3.39
```

Notice that the last case is much slower due to the number of system calls being made. The block size is a factor of 1000 times less than in the first case causing 1000 time more calls to the `read()` and `write()` system calls. Try a `sar -c 2 10` in another window while the test is being run to see this effect.

None of these effects are anything to do with physical disk I/O as the whole `vmunix` file is coming from buffer cache. Prove this to yourself with a `sar -b 2 10` while the test is being run. Notice the 100% read cache hit rate.

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"? _____

Answer Around 400-500 on our test systems

```
# sar -c 2 2    (rp2430)
```

```
HP-UX r206c42 B.11.11 U 9000/800    03/16/04
```

	scall/s	sread/s	swrit/s	fork/s	exec/s	rchar/s	wchar/s
11:18:56							
11:18:58	602	3	1	0.00	0.00	203272	8151
11:19:00	264	4	1	0.00	0.00	4096	512
Average	434	3	1	0.00	0.00	103741	4341

```
# sar -c 2 2    (rx2600)
```

```
HP-UX r265c145 B.11.23 U ia64    04/06/04
```

	scall/s	sread/s	swrit/s	fork/s	exec/s	rchar/s	wchar/s
10:57:02							
10:57:04	719	3	1	0.00	0.00	260840	0
10:57:06	434	3	1	0.00	0.00	4096	4096

Solutions

```
Average      577          3          1      0.00      0.00   132668    2043
```

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
```

Answer Around 20000-30000 on our test systems

```
# sar -c 10 4 (rp2430)
```

```
HP-UX r206c42 B.11.11 U 9000/800 03/16/04
```

	scall/s	sread/s	swrit/s	fork/s	exec/s	rchar/s	wchar/s
11:19:43	17423	3112	1158	130.07	130.07	29710218	147104
11:19:53	12420	3577	2627	63.40	63.40	32159540	8192
11:20:03	23240	4227	1337	192.60	192.60	39581900	17818
11:20:13	26279	3884	700	212.10	212.00	40309248	134963
Average	19840	3700	1456	149.54	149.51	35438766	77037

```
# sar -c 10 4 (rx2600)
```

```
HP-UX r265c145 B.11.23 U ia64 04/06/04
```

	scall/s	sread/s	swrit/s	fork/s	exec/s	rchar/s	wchar/s
11:02:40	39624	4530	1619	290.51	290.51	92426384	77746
11:02:50	28069	5618	3883	171.70	171.60	69435392	80282
11:03:00	27178	5214	3320	189.40	189.40	67771592	62259
11:03:10	31592	5057	2814	222.70	222.60	72799840	91750
Average	31618	5105	2909	218.60	218.55	75612445	78009

- What system calls are generated by **filestress**?

Answer **read()** and **write()**.

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?

Answer Syscall rate is higher than with **filestress**. Non-blocking system calls
Produce rates up to 138,000 per second on an rp2430 and up to 290,000
on an rx2600.

```
# sar -c 10 4 (rp2430)
```


Solutions

HP-UX r206c42 B.11.11 U 9000/800 03/16/04

	scall/s	sread/s	swrit/s	fork/s	exec/s	rchar/s	wchar/s
11:36:11	137619	2	0	0.00	0.00	42863	3376
11:36:21	136788	2	0	0.00	0.00	4506	1946
11:36:31	137887	2	0	0.00	0.00	5734	3277
11:36:41	138224	2	0	0.00	0.00	3686	1229
Average	137629	2	0	0.00	0.00	14171	2457

sar -c 10 4 (rx2600)

HP-UX r265c145 B.11.23 U ia64 04/06/04

	scall/s	sread/s	swrit/s	fork/s	exec/s	rchar/s	wchar/s
11:15:51	287322	27	1	0.50	0.40	60560	4092
11:16:01	288439	7	1	0.00	0.00	233472	20480
11:16:11	289239	9	1	0.00	0.00	27853	4096
11:16:21	290331	4	0	0.00	0.00	14746	3277
Average	288832	12	1	0.12	0.10	84104	7985

The **syscall** program uses the `open()` and `close()` system calls and does no I/O as such. These system calls do not block the process which turns into a CPU hog, only blocking on Priority in the **glance** Wait States page.

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 _____

Answer

sar -w 2 2 (rp2430)

HP-UX r206c42 B.11.11 U 9000/800 03/16/04

	swpin/s	bswin/s	swpot/s	bswot/s	pswch/s
11:39:27	0.00	0.0	0.00	0.0	86
11:39:29	0.00	0.0	0.00	0.0	83
11:39:31	0.00	0.0	0.00	0.0	85
Average	0.00	0.0	0.00	0.0	85

./cs &

sar -w 2 2

HP-UX r206c42 B.11.11 U 9000/800 03/16/04

	swpin/s	bswin/s	swpot/s	bswot/s	pswch/s
11:41:43	0.00	0.0	0.00	0.0	47733
11:41:45	0.00	0.0	0.00	0.0	47733

```
11:41:47      0.00      0.0      0.00      0.0      47471
```

```
Average      0.00      0.0      0.00      0.0      47602
```

```
# sar -w 2 2      (rx2600)
```

```
HP-UX r265c145 B.11.23 U ia64      04/06/04
```

```
11:22:07 swpin/s bswin/s swpot/s bswot/s pswch/s
```

```
11:22:09      0.00      0.0      0.00      0.0      150
```

```
11:22:11      0.00      0.0      0.00      0.0      177
```

```
Average      0.00      0.0      0.00      0.0      164
```

```
# ./cs&
```

```
# sar -w 2 2
```

```
HP-UX r265c145 B.11.23 U ia64      04/06/04
```

```
11:22:57 swpin/s bswin/s swpot/s bswot/s pswch/s
```

```
11:22:59      0.00      0.0      0.00      0.0      81912
```

```
11:23:01      0.00      0.0      0.00      0.0      82728
```

```
Average      0.00      0.0      0.00      0.0      82319
```

Notice that we go from an idle context switch rate (**pswch/s**) of approx 100 processes per second up to 47000 or 82000!

Additionally, you can look at the **glance** CPU Report (c). Note how much of the CPU time is spent doing context switching. (About 15%)

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 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? _____

Answer At or near 100%

What are the **nice** values of the processes receiving the most CPU time? _____

Answer 10

What is the average number of jobs in the CPU run queue? _____

Answer Varies with configuration – should be approx 3-5

```
# uptime
12:05pm up 4 days, 19:38, 7 users, load average: 4.73, 3.31, 2.26
#
```

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.

Glance global (g) page output (rp2430):

PROCESS LIST							Users=		1
Process Name	PID	PPID	Pri	User Name	CPU Util (100% max)	Cum CPU	Disk IO Rate	RSS	Thd Cnt
proc8	27425	1	215	root	50.1/49.4	138.6	0.0/ 0.0	168kb	1
proc3	27420	1	221	root	48.4/49.2	138.0	0.0/ 0.0	168kb	1
prm3d	1462	1	168	root	0.0/ 0.2	1125.1	0.0/ 0.0	26.6mb	19
proc5	27422	1	168	root	0.0/ 0.2	0.5	4.0/ 4.0	168kb	1


```
proc2      27419      1 168 root      0.0/ 0.2      0.5  3.8/ 3.9   168kb      1
```

Glance global (g) page output (rx2600):

PROCESS LIST							Users=		1
Process Name	PID	PPID	Pri	User Name	CPU Util (100% max)	Cum CPU	Disk IO Rate	RSS	Thd Cnt
proc3	26194	1	219	root	50.8/49.3	81.5	0.0/ 0.0	268kb	1
proc8	26199	1	216	root	48.5/49.4	81.6	0.0/ 0.0	268kb	1
scopeux	2105	1	127	root	0.0/ 0.0	13.3	0.0/ 0.0	20.7mb	1
prm3d	2139	1	168	root	0.0/ 0.1	77.5	0.0/ 0.0	49.5mb	19
ia64_corehw	2989	1	154	root	0.0/ 0.1	65.9	1.1/ 0.0	1.8mb	1
proc2	26193	1	168	root	0.0/ 0.1	0.2	7.6/ 7.7	256kb	1
proc5	26196	1	168	root	0.0/ 0.1	0.2	7.6/ 5.8	256kb	1

proc3 and **proc8** are the main CPU hogs. They have been run with nice values of 10! The process pair are accounting for almost 100% of the CPU between them. With the same CPU rates and RSS (Resident Set Size), it is likely that these are identical programs. Selecting one of these processes in **glance** reveals no disc I/O and a context switch profile which is always forced.

proc5 and **proc2** also manage to execute with 0.2% CPU utilization each. Again these look like a pair. If you select one of these programs and look at the Process Resource page you can see a small amount of write disk I/O, most of which is logical. The main Wait Reason for this process is **SLEEP**. It would appear that these processes do a small amount of disk I/O and then call `sleep()` and pause for some time intentionally.

proc1 and **proc7** are a pair. On selecting one of these we see a nice value of 39! These processes find it nearly impossible to get CPU with the real time pair of **proc3** and **proc8** taking all the CPU resource. If you watch the Dispatches metric on the Process Resource page they can be seen to get one or two slices of CPU very infrequently. You should also see that for every **Dispatch** (these are rare), there is always an accompanying **Forced Cswitch**. You can conclude that these processes would be CPU hogs if they were not so crippled by their own high nice values and the aggression of **proc3** and **proc8**.

proc4 and **proc6** are the last pair. They have standard nice values of 20 and seem to do nothing but call the `sleep()` system call. They are being dispatched slightly more frequently than **proc1** and **proc7** and they are always subject to **Voluntary CSwitch**. These processes are not CPU hogs. They also do no disk I/O of any kind.

None of the above processes had any significant memory size.

- 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? _____

Answer:

```
# timex /home/h4262/baseline/short &      (rp2430)
The last prime number is : 49999
```

Solutions

```
real      56.44
user      10.66
sys       0.01
```

```
# timex /home/h4262/baseline/short & (rx2600)
```

```
# The last prime number is : 99991
```

```
real      1:02.38
user       8.48
sys       0.00
```

6. Compare your results to the baseline established in the lab exercise in module 1, step 7.

Answer: Total execution time is over 5 times slower!

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? _____

Answer: Should be greater than 50%. (the more, the merrier!)

Output of `rp2430 glance (g)` page below

PROCESS LIST										Users=	1
Process Name	PID	PPID	Pri	User Name	CPU Util (100% max)	Cum CPU	Disk IO Rate	RSS	Thd Cnt		
proc2	27761	1	1	root	92.0/92.3	723.2	0.0/ 0.0	168kb	1		
prm3d	1462	1	168	root	0.0/ 0.2	1137.2	0.0/ 0.0	26.6mb	19		

Output of `rp2430 glance (a)` page below

CPU BY PROCESSOR							Users=	1
CPU	State	Util	LoadAvg(1/5/15 min)			CSwitch	Last Pid	
0	Enable	93.2	0.5/	0.6/	1.7	1724	27761	

Output of `rx2600 glance (g)` page below

PROCESS LIST										Users=	1
Process Name	PID	PPID	Pri	User Name	CPU Util (100% max)	Cum CPU	Disk IO Rate	RSS	Thd Cnt		
proc2	26469	1	1	root	71.5/71.6	47.1	0.0/ 0.0	288kb	1		
prm3d	1462	1	168	root	0.0/ 0.2	1137.2	0.0/ 0.0	26.6mb	19		

Output of `rx2600 glance (a)` page below

CPU BY PROCESSOR							Users=	1
CPU	State	Util	LoadAvg(1/5/15 min)			CSwitch	Last Pid	
0	Enable	73.1	0.0/	0.2/	0.8	1432	26469	

What does **sar** report for CPU utilization? _____

Solutions

Answer: **sar** reports the CPU is mostly idle. Util is less than 10%.

```
# sar -u 5 200
```

```
HP-UX r206c42 B.11.11 U 9000/800 03/16/04
```

	%usr	%sys	%wio	%idle
13:45:58				
13:46:03	4	2	0	94
13:46:08	0	1	0	99
13:46:13	1	1	0	98
13:46:18	0	0	0	100
13:46:23	1	1	0	98

This is very strange; the tools totally disagree with each other. **sar** is reporting over 90% idle with **glance** reporting over 80% busy! They cannot both be right. Which one do you trust?

The output of **top** is also confused. It sees the busy process but still reports 90% idle!

```
Load averages: 0.50, 0.56, 1.41 (rp2430)
```

```
112 processes: 99 sleeping, 13 running
```

```
Cpu states:
```

LOAD	USER	NICE	SYS	IDLE	BLOCK	SWAIT	INTR	SSYS
0.50	0.6%	0.0%	2.2%	97.2%	0.0%	0.0%	0.0%	0.0%

```
Memory: 91236K (64076K) real, 365020K (299140K) virtual, 30120K free Page# 1/8
```

TTY	PID	USERNAME	PRI	NI	SIZE	RES	STATE	TIME	%WCPU	%CPU	COMMAND
pts/tb	27761	root	1	20	1664K	148K	sleep	14:49	92.56	92.40	proc2

```
Load averages: 0.03, 0.12, 0.68 (rx2600)
```

```
128 processes: 107 sleeping, 20 running, 1 zombie
```

```
Cpu states:
```

LOAD	USER	NICE	SYS	IDLE	BLOCK	SWAIT	INTR	SSYS
0.03	0.2%	0.0%	0.0%	99.8%	0.0%	0.0%	0.0%	0.0%

```
Memory: 197664K (154768K) real, 608492K (523032K) virtual, 23516K free Page# 1/10
```

TTY	PID	USERNAME	PRI	NI	SIZE	RES	STATE	TIME	%WCPU	%CPU	COMMAND
ttylp0	26469	root	1	20	3304K	252K	sleep	4:08	71.77	71.64	proc2

What is the priority of the process receiving the most CPU time? _____

Answer

The **proc2** process is the culprit and is running with the high UNIX real time priority of 1.

How much time is the process spending in the **sigpause** system call? _____

Answer

Now this is where the clues start!

The Wait States for `proc2` show that it is blocked on **SLEEP** when it is not running. This wait state is the result of the process putting itself to sleep.

To see the system calls that the process is calling hit the **F6** softkey or **L** key once you have selected the process. `glance` will collect the data and present it after about 10-20 seconds.

rp2430:

```
System Calls PID: 27761, proc2      PPID:      1  euid:      0  User: root
System Call Name      ID      Count   Rate      Elapsed      Elapsed
                        Time      Cum Ct    CumRate    CumTime
-----
sigpause              111      449    99.7      0.35218      1497      74.1      1.17095
sigcleanup            139      450   100.0      0.00166      1500      74.2      0.00553
```

rx2600:

```
System Calls PID: 26469, proc2      PPID:      1  euid:      0  User: root
System Call Name      ID      Count   Rate      Elapsed      Elapsed
                        Time      Cum Ct    CumRate    CumTime
-----
sigpause              111      525   100.9      1.49255      1500      74.2      4.26847
sigcleanup            139      525   100.9      0.00143      1500      74.2      0.00408
```

The `sigpause()` call is causing the sleep blocks that we see in the Wait States page. The interesting thing is that the rate at which the program calls `sigpause()` is always 100 times per second. That is 10 ms (milli-seconds) between calls.

How can a program be so coordinated with the wall clock and what is it using to achieve this synchronization? Can you tell what it is yet?

How is the process being context switched (forced or voluntary)? _____

Answer

Review the Resource Summary page again for `proc2` and you will see that all the context switches are Voluntary. This is not the expected case for a CPU hog. How is it that a process can use so much CPU and never be seen by the scheduler and thrown off the CPU?

The Bottom Line

If you examine the code of the lab you will see that the process arms a trap waiting for the system hardware clock (the tick) to pop. When this occurs the program wakes up and wastes CPU for an amount of time that your instructor has tuned to be just under 10ms (see `waste.c`). The program then arms the trap again and voluntarily goes to sleep waiting for the next hardware tick.

Remember the UNIX scheduler analyzes system activity on the hardware tick intervals and our program has done a good job at never being around at these times! It's a free lunch.

Solutions

The standard UNIX tools (**sar** and **top** for example) feed on the scheduler's internal statistics for measurement data and so they get the wrong story.

glance however uses the **midaemon**, which recalculates performance stats every time a process returns from a system call. And you cannot play this game without system calls.

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? _____

Answer: (rp2430)

```
# timex /home/h4262/baseline/short &
The last prime number is : 49999
```

```
real    2:32.86
user      10.88
sys       0.07
```

(rx2600)

```
# timex /home/h4262/baseline/short &
```

```
# The last prime number is : 99991
```

```
real    30.86
user      8.51
sys       0.01
```

Our old benchmark figure was around 10 seconds (real) so this is significantly slower. This program is running in the gaps that the **proc2** process is leaving. You could further modify **waste.c** to use more of the tick period.

5. End the CPU load by executing the **KILLIT** script.

```
# ./KILLIT
```


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.

The following solution data came from an rp2430 server with 640MB of physical memory and 2GB of device swap, and an rx2600 server with 1012MB of physical memory and 2GB of device swap. The rp2430 was running HP-UX 11i v1 and the rx2600 was running 11i v2.

2. Before starting the background processes, look up the current value for `maxdsiz` using the `kmtune` command on 11i v1 and the `kctune` command on 11i v2.

On the rp2430:

```
# kmtune -lq maxdsiz
```

Answer:

Varies with configuration – probably 64MB if you are pre 11i and 256MB for 11i v1.

```
# kmtune -lq maxdsiz
Parameter:      maxdsiz
Current: 0x10000000
Pending: 0x10000000
Default: 0x10000000
Minimum: -
Module: -
Version -
Dynamic: No
#
```

The number is in hex... Converting this to decimal = 268435456 = 256MB

On the rx2600:

```
# kctune -avq maxdsiz
```

Answer:

Varies with configuration – probably 1GB for 11i v2.

```
# kctune -avq maxdsiz
Tunable          maxdsiz
```

Solutions

Description Maximum size of the data segment of a 32-bit process (bytes)

```
Module                      vm
Current Value               1073741824 [Default]
Value at Next Boot         1073741824 [Default]
Value at Last Boot         1073741824
Default Value               1073741824
Constraints                  maxdsiz >= 262144
                              maxdsiz <= 4294963200
Can Change                  Immediately or at Next Boot
```

The number is in decimal = 1073741824 = 1GB

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:

```
# kctune 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?

rp2430:

```
# vmstat 2 2
procs                      memory                      page
faults                      cpu
r      b      w      avm      free      re      at      pi      po      fr      de      sr      in
sy      cs      us sy id
3      0      0      75182      92519      3      0      0      0      0      0      0      104
408      138      1      0 99
3      0      0      75182      92465      3      0      1      0      0      0      0      106
214      75      0      0 100
```

We have around 92000 free pages which equates to 368MB.

rx2600:

```
# vmstat 2 2
procs                      memory                      page
faults                      cpu
r      b      w      avm      free      re      at      pi      po      fr      de      sr      in
sy      cs      us sy id
2      0      0      124095      97927      466      165      298      0      0      0      2      1134
```

```

47856  476  14 19 67
      2    0    0 124095  96427 137  26  69  0  0  0  26  536
21509  470  3  3 94

```

We have around 97000 free pages which equates to 388MB.

- Use the **RUN** script to start the background processes:

```
# ./RUN
```

- 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.

Go for the **m** page of **glance** for the best info. You have to be quick off the mark after starting the leak programs!

MEMORY REPORT						Users=	1
Event	Current	Cumulative	Current Rate	Cum Rate	High Rate		
Page Faults	588	1301	113.0	116.1	137.1		
Page In	1	33	0.1	2.9	6.1		
Page Out	0	0	0.0	0.0	0.0		
KB Paged In	0kb	36kb	0.0	3.2	6.9		
KB Paged Out	0kb	0kb	0.0	0.0	0.0		
Reactivations	0	0	0.0	0.0	0.0		
Deactivations	0	0	0.0	0.0	0.0		
KB Deactivated	0kb	0kb	0.0	0.0	0.0		
VM Reads	0	3	0.0	0.2	0.5		
VM Writes	0	0	0.0	0.0	0.0		
Total VM :	384.9mb	Sys Mem :	182.3mb	User Mem:	96.9mb	Phys Mem:	640.0mb
Active VM:	342.1mb	Buf Cache:	32.4mb	Free Mem:	328.4mb		

- What is the current amount of free memory?

Answer:Varies with configuration
Already this has dropped to 328.4MB

- What is the size of the buffer cache?

Answer:Varies with configuration
In our case this is 32.4MB

- Is there any paging to the swap space?

Answer:Varies with configuration
No not in the last sample, see **KB paged Out** above

- How much swap space is currently reserved?

Answer:Varies with configuration
Get this from **swapinfo**. Again you need to do this just after the programs start:
In our case around 249MB.

```
# swapinfo -tm
```


Solutions

	Mb	Mb	Mb	PCT	START/	Mb		
TYPE	AVAIL	USED	FREE	USED	LIMIT	RESERVE	PRI	NAME
dev	2048	0	2048	0%	0	-	1	
/dev/vg00/lvol2								
reserve	-	379	-379					
memory	1013	330	683	33%				
total	3061	709	2352	23%	-	0	-	

The total swap space "used" (used = really used + reserved) is the figure in bold. More detail on swap management is in Module 7. For now take the bottom line figure in bold above.

- Which process has the largest Resident Set Size (RSS)?

Answer proc1. You can see that from the global process list in **g**lance (the **g** key). As you watch it, it will grow until vhand kicks in and limits its RSS. However, the VSS will continue to grow. Select that process (with **s**) and observe to RSS/VSS figure.

PROCESS LIST									
Process Name	PID	PPID	Pri	User	CPU Util (100% max)	Cum CPU	Disk IO Rate	Users= RSS	Thd Cnt
proc1	3267	1	168	root	0.0/ 0.2	1.0	0.0/ 0.0	275.8mb	1
proc2	3268	1	168	root	0.0/ 0.1	0.4	0.0/ 0.0	114.6mb	1
proc3	3269	1	168	root	0.0/ 0.0	0.2	0.0/ 0.0	56.7mb	1
proc4	3270	1	168	root	0.0/ 0.0	0.1	0.0/ 0.0	27.7mb	1
alarmgen	3277	3276	168	root	0.0/ 0.0	0.1	1.3/ 0.1	1.6mb	6
vhand	2	0	128	root	0.4/ 0.2	2.0	81.7/44.2	64kb	1

Resources PID: 3267, proc1		PPID: 1		euid: 0		User: root	
CPU Usage (util):	0.0	Log Reads :	0	Wait Reason :	SLEEP		
User/Nice/RT CPU:	0.0	Log Writes:	0	Total RSS/VSS :	275.7mb/479.1mb		
System CPU :	0.0	Phy Reads :	0	Traps / Vfaults:	0/	542	
Interrupt CPU :	0.0	Phy Writes:	0	Faults Mem/Disk:	0/	0	
Cont Switch CPU :	0.0	FS Reads :	0	Deactivations :	0		
Scheduler :	HPUX	FS Writes :	0	Forks & Vforks :	0		
Priority :	168	VM Reads :	0	Signals Recd :	0		
Nice Value :	20	VM Writes :	0	Mesg Sent/Recd :	0/	0	
Dispatches :	5	Sys Reads :	0	Other Log Rd/Wt:	0/	0	
Forced CSwitch :	0	Sys Writes:	0	Other Phy Rd/Wt:	0/	0	
VoluntaryCSwitch:	5	Raw Reads :	0	Proc Start Time			
Running CPU :	0	Raw Writes:	0	Tue Apr 6 14:29:16 2004			
CPU Switches :	0	Bytes Xfer:	0kb	:			

- What is the data segment size of the process with the largest RSS?

Answer:select the memory regions page for **proc1** with the **M** key.

Memory Regions PID: 3267, proc1				PPID: 1	euid: 0	User: root
Type	RefCt	RSS	VSS	Locked	File Name	
NULLDR/Shared	87	4kb	4kb	0kb	<nullldref>	
TEXT /Shared	2	4kb	4kb	0kb	/home/.../leak/proc1	
DATA /Priv	1	301.0mb	716.2mb	0kb	/home/.../leak/proc1	
MEMMAP/Priv	1	0kb	16kb	0kb	/usr/lib/tztab	

```

MEMMAP/Priv      1      4kb      4kb      0kb <mmap>
MEMMAP/Priv      1      4kb      8kb      0kb <mmap>
MEMMAP/Priv      1      0kb      8kb      0kb <mmap>
MEMMAP/Priv      1     24kb     28kb      0kb /usr/lib/hpux32/libc.so.
MEMMAP/Priv      1     40kb     40kb      0kb <mmap>

Text  RSS/VSS:  4kb/  4kb  Data  RSS/VSS:301mb/716mb  Stack RSS/VSS:  4kb/  8kb
Shmem RSS/VSS:  0kb/  0kb  Other RSS/VSS:1.6mb/3.2mb

```

The data segment size in this example is 301/716 MB and growing!

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?

Answer This is going to take several minutes. The **maxdsiz** limit is probably either 256MB or 1GB on the test system. Be careful! **maxdsiz** is a limit on the VSS (Virtual Set Size) and not the RSS (Resident Set Size).

System starts doing a LOT of disk I/O. Look for the large “F” bar in the Disc Util global meter.

- Why does disk utilization become so high at this point?

Answer Kernel is dumping the **core** file of the user process in our case. You will probably run out of disc space in the **/home** file system. You may want to remove the **/home/h4262/memory/leak/core** file! Remember it is not the process that is doing the disk I/O, it is the kernel that is doing it to produce the **core** file.

6. As the other processes grow towards their maximum data segment size, continue to monitor the following:

- Free memory

```

# vmstat 2 2
procs
 faults      cpu
 r    b    w  avm   free  re  at  pi  po  fr  de  sr  in
 sy  cs  us sy id
 2    0    0 321403 91118  54  19  79 285  16  0 359 548
4962 326  2  3 95
 2    0    0 321403 90413  1   0 115  12  0  0  0 397
552  191  0  0 100

```

Not a lot of free memory now. The system is under memory pressure and is paging out to stabilize the memory system

- Swap space reserved

```

# swapinfo -tm
      Mb      Mb      Mb  PCT  START/      Mb
TYPE    AVAIL    USED    FREE  USED  LIMIT RESERVE  PRI  NAME
dev      2048     715    1333  35%    0      -    1  /dev/vg00/lvol2

```

Solutions

```

reserve      -      341    -341
memory      1013      340      673    34%
total       3061     1396     1665    46%    -      0    -

```

Swapspace is up to 46% utilization!

- The size of the processes' data segments

All the **proc (n)** processes continue to grow (see VSS) just like **proc1** did and they are aborted in the same way when they cross the line (**maxdsiz**).

- The RSS of the processes

The running memory hog processes compete for the limited real memory resource. We didn't have a lot free at the start of the test and the lab procs all want to grow to the **maxdsiz** limit. They cannot all fit together so they fight. This is a classic memory thrash situation.

- The number of page-outs/page-ins to the swap space

This depends on when you look! These figures were taken while **proc2** was still on the move and free memory was approaching its minimum.

```

# vmstat 2 10
procs      memory      page
faults      cpu      avm      free      re      at      pi      po      fr      de      sr      in
r      b      w      us      sy      id
2      0      0      166464      2692      0      0      0      0      0      0      0      103
173      82      0      0      100
2      1      0      170444      1649      0      0      13      0      0      0      0      123
209      92      0      0      100
2      1      0      170444      1028      0      0      8      5      4      0      1256      122
189      88      0      6      94
2      1      0      170444      1146      8      0      6      101      109      0      9869      225
176      129      0      5      95
2      1      0      170444      1392      12      0      5      263      69      0      9659      316
175      112      0      0      100
2      1      0      170444      1366      12      0      5      312      44      0      8186      331
190      156      0      0      100
1      0      0      169455      1090      9      0      5      304      28      0      6410      316
209      201      0      0      100
1      0      0      169455      1112      6      0      3      351      31      0      5334      359
193      163      0      1      99
1      0      0      169455      1048      3      0      2      332      19      0      3902      339
180      133      5      0      95
1      0      0      169455      1600      5      0      0      396      12      0      2576      370
240      119      0      4      96

```

7. Run the two baseline programs, **short** and **diskread**.

```

# timex /home/h4262/baseline/short
# timex /home/h4262/baseline/diskread

```


rp2430:

```
# timex /home/h4262/baseline/short
The last prime number is : 49999
```

```
real      12.00
user      10.86
sys       0.02
```

```
# timex /home/h4262/baseline/diskread
DiskRead: System      : [HP-UX]
DiskRead: RawDisk     : [/dev/rdisk/clt15d0]
DiskRead: Start reading : 1024MB
1024+0 records in
1024+0 records out
```

```
real      31.79
user      0.02
sys       0.53
```

rx2600:

```
# timex /home/h4262/baseline/short &
```

```
# The last prime number is : 99991
```

```
real      8.54
user      8.48
sys       0.00
```

```
# timex /home/h4262/baseline/diskread &
[1]      3841
```

```
root@r265c145:/home/h4262/memory/leak
# DiskRead: System      : [HP-UX]
DiskRead: RawDisk     : [/dev/rdisk/c2t1d0s2]
DiskRead: Start reading : 2048MB
2048+0 records in
2048+0 records out
```

```
real      29.60
user      0.01
sys       0.16
```

How does the performance of these programs compare to their earlier runs?

Answer: **short** takes a little longer. The CPU is not under much pressure at this time so compute bound processes will not be affected (unless they need memory!). It is a different story for **diskread**, in the first test case, it took noticeably longer due to the disk load already in progress for the paging activity. It is not good to have swap space on your application disks!

8. When finished monitoring the behavior of processes with memory leaks, clean up the processes.

Solutions

- 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	

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 -m` 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	512	0
reserve	-	139
memory	451	27

Answer Varies with configuration, examples below.

```
# swapinfo -m (rp2430)
      Mb      Mb      Mb  PCT  START/      Mb
TYPE  AVAIL  USED  FREE  USED  LIMIT RESERVE  PRI  NAME
dev    512    0    512   0%    0      -    1  /dev/vg00/lvol2
reserve -    139  -139   6%
memory 451    27   424   6%
```

```
# swapinfo -m (rx2600)
      Mb      Mb      Mb  PCT  START/      Mb
TYPE  AVAIL  USED  FREE  USED  LIMIT RESERVE  PRI  NAME
dev   2048    75  1973   4%    0      -    1  /dev/vg00/lvol2
reserve -    189  -189   33%
memory 1013   339   674   33%
```

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)?

Answer Varies with configuration, in our case it is 964 Mb or 3 Gb
(as seen in the bolded figures below.)

Solutions

```
# swapinfo -tm (rp2430)
      Mb      Mb      Mb      PCT      START/      Mb
TYPE   AVAIL    USED    FREE  USED  LIMIT RESERVE  PRI  NAME
dev     512       0     512    0%      0      -     1  /dev/vg00/lvol2
reserve -     139    -139
memory  451       27     424    6%
total   963    166    797   17%      -      0     -
```

```
# swapinfo -mt (rx2600)
      Mb      Mb      Mb      PCT      START/      Mb
TYPE   AVAIL    USED    FREE  USED  LIMIT RESERVE  PRI  NAME
dev    2048       74    1974    4%      0      -     1  /dev/vg00/lvol2
reserve -     190    -190
memory 1013      339     674   33%
total   3061    603    2458   20%      -      0     -
```

What is the total space "reserved"?

Answer Varies with configuration. Swap space is first reserved and then it may (or may not) be used by the process that reserved it. The bottom line is that reserved swap space is no more available than used swap space so the only figure that really matters here are the totals underlined (166 Mb and 603 Mb). This figure is unavailable to any other process.

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?

Answer It should and it does. But you have to be careful when you look.

It is easy for some other activity on the system to "spoil" the results. You may want to try it 2 or 3 times to see if your results change. What SHOULD happen is that the "reserve-USED" entries should increase and then decrease by exactly the same amount.

rp2430:

```
# swapinfo
      Kb      Kb      Kb      PCT      START/      Kb
TYPE   AVAIL    USED    FREE  USED  LIMIT RESERVE  PRI  NAME
dev    524288       0    524288    0%      0      -     1  /dev/vg00/lvol2
reserve - 1444444 -1444444
memory 462248    28384    433864    6%
```

```
# sh
```

```
# swapinfo
      Kb      Kb      Kb      PCT      START/      Kb
TYPE   AVAIL    USED    FREE  USED  LIMIT RESERVE  PRI  NAME
dev    524288       0    524288    0%      0      -     1  /dev/vg00/lvol2
reserve - 144768 -144768
```

```
memory 462248 28384 433864 6%
# exit

# swapinfo
      Kb      Kb      Kb  PCT  START/      Kb
TYPE  AVAIL  USED  FREE  USED  LIMIT  RESERVE  PRI  NAME
dev    524288    0 524288   0%    0    -    1 /dev/vg00/lvol2
reserve - 144444 -144444
memory 462248 28388 433860   6%
```

rx2600:

```
# swapinfo
      Kb      Kb      Kb  PCT  START/      Kb
TYPE  AVAIL  USED  FREE  USED  LIMIT  RESERVE  PRI  NAME
dev    2097152 75652 2021500   4%    0    -    1 /dev/vg00/lvol2
reserve - 194900 -194900
memory 1037064 346740 690324  33%
```

sh

```
# swapinfo
      Kb      Kb      Kb  PCT  START/      Kb
TYPE  AVAIL  USED  FREE  USED  LIMIT  RESERVE  PRI  NAME
dev    2097152 75652 2021500   4%    0    -    1 /dev/vg00/lvol2
reserve - 195540 -195540
memory 1037064 346740 690324  33%
```

exit

```
# swapinfo
      Kb      Kb      Kb  PCT  START/      Kb
TYPE  AVAIL  USED  FREE  USED  LIMIT  RESERVE  PRI  NAME
dev    2097152 75652 2021500   4%    0    -    1 /dev/vg00/lvol2
reserve - 194900 -194900
memory 1037064 346740 690324  33%
```

If you see that some swap was reserved and not released, then there is something else going on in the background that is skewing the figures.

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 Util. percentage increases in **glance**. Type:

```
# /home/h4262/memory/paging/mem256 &
```

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 Util. 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?

Answer Varies with configuration, depends on your swap space.

Solutions

On the rp2430, after 12 copies of **mem256** the test system swap space was almost gone. Below is what happened when the 13th process was introduced.

```
# swapinfo -tm
      Mb      Mb      Mb  PCT  START/      Mb
TYPE   AVAIL   USED   FREE  USED  LIMIT RESERVE  PRI  NAME
dev      512    461     51   90%     0      -    1  /dev/vg00/lvol2
reserve    -     51    -51
memory   451    399     52   88%
total    963    911     52   95%      -      0    -
```

```
# /home/h4262/memory/paging/mem256&
[13] 2864
```

```
# exec(2): insufficient swap or memory available.
```

```
[13] + Done(9) /home/h4262/memory/paging/mem256&
```

On the rx2600, after 37 copies of **mem256** the test system swap space was almost gone. Below is what happened when the 38th process was introduced.

```
# swapinfo -tm
      Mb      Mb      Mb  PCT  START/      Mb
TYPE   AVAIL   USED   FREE  USED  LIMIT RESERVE  PRI  NAME
dev    2048    1978     70   97%     0      -    1  /dev/vg00/lvol2
reserve    -     70    -70
memory  1013     991     22   98%
total   3061    3039     22   99%      -      0    -
```

```
# ./mem256&
[38] 4159
exec(2): insufficient swap or memory available.
```

What prevented an additional **mem256** process from being started?

Answer "Insufficient swap or memory available"

Kill all **mem256** processes to restore performance.

5. Recompile the kernel, disabling **pseudo-swap**. Use the following procedure:

11i v1 and earlier:

```
# cd /stand/build
# /usr/sbin/sysadm/system_prep -s system
# echo "swapmem_on 0" >> system
# mk_kernel -s system
# cd /
# shutdown -ry 0
```

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.

rp2430:

```
Press any key to interrupt the boot process
Main menu> boot pri isl
Interact with IPL> y
ISL> hpux (;0)/stand/build/vmunix_test
```

rx2600:

(Nothing special needs to be done)

7. Once the system reboots, login and execute **swapinfo**.

Is there a memory entry? Why or why not?

Answer No. Pseudo-swap has been disabled.

Will the same number of **mem256** processes be able to execute as earlier?

Answer No.

How many **mem256** processes can be started now?

Answer Varies with configuration

On the rp2430, only 6 processes could be started successfully. On the rx2600, only 27 processes could be started successfully.

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.**

Solutions

If you did not add the second disk earlier,

```
# vgdisplay -v | grep Name (Note the physical disks used by vg00)
# ioscan -fnC disk (Note which disk is unused by LVM)
# pvcreate -f <raw_dev_file_of_second_disk>
# vgextend /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 was 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
```

Answer:

↳ de préférence à deux

```
# swapinfo -tm
```

	Mb	Mb	Mb	PCT	START/	Mb		
TYPE	AVAIL	USED	FREE	USED	LIMIT	RESERVE	PRI	NAME
dev	512	0	512	0%	0	-	1	/dev/vg00/lvol2
reserve	-	130	-130					
total	512	130	382	25%	-	0	-	

```
# 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 (if necessary), to increase `maxswapchunks`. Use the following procedure:

11i v1 and earlier (ONLY!)

```
# 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.

11i v1 and earlier (ONLY!)

```
Press any key to interrupt the boot process
Main menu> boot pri isl
Interact with IPL> y
ISL> hpx (/;0)/stand/build/vmunix_test
```

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 (rp2430)
      Mb      Mb      Mb  PCT  START/      Mb
TYPE   AVAIL   USED   FREE  USED  LIMIT RESERVE  PRI  NAME
dev      512      0     512   0%      0      -    1  /dev/vg00/lvol2
dev      512      0     512   0%      0      -    1  /dev/vg00/swap1
reserve  -     141   -141
total   1024     141     883  14%      -      0  -
```

```
# swapinfo -tm (rx2600)
      Mb      Mb      Mb  PCT  START/      Mb
TYPE   AVAIL   USED   FREE  USED  LIMIT RESERVE  PRI  NAME
dev    2048      86   1962   4%      0      -    1  /dev/vg00/lvol2
dev    2048      0   2048   0%      0      -    1  /dev/vg00/swap1
reserve  -     158   -158
total   4096     244   3852   6%      -      0  -
```

Done!

11. Start enough **mem256** processes to make the system start paging.

Answer: This depends on how much memory you have but on an rp2430 with 640MB, I found that 8 processes got things paging nicely! On an rx2600, 10 should do nicely.

```
# vmstat 2 2
procs
faults      cpu      memory      page
r  b  w  avm  free  re  at  pi  po  fr  de  sr  in
```


Solutions

```

sy      cs  us sy id
9        0    0 180106 5064 34 0 192 340 99 0 3136 339
213     471 100 0 0
9        0    0 180106 5056 23 0 122 217 63 0 2006 216
191     355 100 0 0

```

Note the system is paging constantly in the `vmstat` output and free memory is very low.

12. Measure the disk I/O to see what is happening with swap space. **Go to question 15 when you have finished.**

Answer: The I/O should be balanced across both disks!

```
# sar -d 5 2      (rp2430)
```

```
HP-UX r206c41 B.11.11 U 9000/800 03/18/04
```

	device	%busy	avque	r+w/s	blks/s	avwait	avserv
14:22:12							
14:22:17	c1t15d0	87.03	24.73	409	12222	33.45	13.86
	c3t15d0	60.68	23.21	406	12093	31.03	9.24
14:22:22	c1t15d0	82.60	22.01	395	12209	28.53	12.26
	c3t15d0	72.20	19.57	385	11976	25.00	10.57
Average	c1t15d0	84.82	23.39	402	12216	31.03	13.08
Average	c3t15d0	66.43	21.43	396	12034	28.10	9.89

```
# sar -d 5 2      (rx2600)
```

```
HP-UX r265c145 B.11.23 U ia64 04/07/04
```

	device	%busy	avque	r+w/s	blks/s	avwait	avserv
11:28:10							
11:28:15	c2t1d0	9.38	0.50	25	542	0.00	6.05
	c2t0d0	3.79	0.50	14	271	0.01	4.71
11:28:20	c2t1d0	21.40	6.75	79	2373	2.85	5.35
	c2t0d0	6.60	10.42	47	1229	3.86	3.94
Average	c2t1d0	15.38	5.25	52	1456	2.16	5.51
Average	c2t0d0	5.19	8.13	31	750	2.97	4.12

This has doubled the effective performance of swap space. The results would be even better if the swap disks were on different controllers.

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

```

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.

List the current amount of swap space in use.

Answer Varies with configuration. Use `swapinfo -tm`.

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 10MB and 10MB.
or
B) The distribution would be 15MB and 5MB.

Answer B. `vhand` does not consider what the previous utilization was.

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 `mem256` processes to make the system start paging.

Answer: This depends on how much memory you have but on a 640MB system I found that 8 processes got things paging nicely!

```
# vmstat 2 2
procs      memory      page
 faults    cpu
  r   b   w   avm   free   re   at   pi   po   fr   de   sr   in
 sy  cs  us sy  id
10   0   0 175597 6489   12   8    2  31   11   0  467    0
271  58  26  4  70
10   0   0 175597 6414   20   0   27  87   22   0 1316  103
300 254 100  0  0
```

Note the system is paging constantly in the `vmstat` output and free memory is very low.

Is the new paging activity being distributed evenly across the paging devices?

Answer No. It is confined to `lv02` (primary swap), `swap1`, and `swap3`.

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:

Solutions

```
# cd /  
# shutdown -ry 0
```

For **11i v2** and later, follow this procedure:

```
# cd /  
# kctune swapmem_on=1  
# shutdown -ry 0
```

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
```

- X 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
```

Solutions

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:
sys:                                Logl Rds:
                                      Phys Rds:
```

Answer:

```
# timex cat file* > /dev/null (rp2430)
```

```
real:    0.73
user:    0.01
sys:     0.11                                Logl Rds:  2560
                                              Phys Rds:   500
```

```
# timex cat file* > /dev/null (rx2600)
```

```
real    0.34
user    0.00
sys     0.06                                Logl Rds:  2560
                                              Phys Rds:  2560
```

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:
sys:                                Logl Rds:
                                      Phys Rds:
```

Answer:

```
# timex cat file* > /dev/null (rp2430)
```

```
real:    0.06
user:    0.01
sys:     0.05                                Logl Rds:  2560
                                              Phys Rds:    0
```

```
# timex cat file* > /dev/null (rx2600)
```

```
real:    0.02
user:    0.00
sys:     0.02                                Logl Rds:  2560
                                              Phys Rds:    0
```

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 VxFS 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?

Answer:

The disk got over 80% busy.

The average number of requests in the I/O queue reached around 53 on the rp2430 and 442 on the rx2600.

The average wait time of a request was around 65 ms on the rp2430 and 182 ms on the rx2600.

The task took around 12.5 seconds on the rp2430 and 7.5 seconds on the rx2600.

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**. Record the results below:

`# ./disk_long` **glance** I/O by Disk Report

Util: Qlen:

Answer:

Utilization reached 86% and queue length reached 55 on the rp2430.

Utilization reached 85% and queue length reached 414 on the rx2600.

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**. Record results below:

`# ./disk_long` **glance** I/O by Disk Report

LogI I/O: Phys I/O:

Solutions

Answer:

Logical I/Os reached 4059 and Physical I/Os reached 806 on the rp2430.
Logical I/Os reached 4702 and Physical I/Os reached 1528 on the rx2600.

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?

9-14. LAB: HFS Performance Issues

Directions

The following lab illustrates a number of performance issues related to HFS file systems.

1. A 512 MB HFS file system is required for this lab. Use the `mount` and `bdf` commands to determine if such a file system is available.

```
# mount -v
# bdf
```

If there is no such HFS file system available, create one using the commands below:

```
# lvcreate -n hfs vg00
# lvextend -L 512 /dev/vg00/hfs /dev/dsk/cXtYdZ (second disk)
# newfs -F hfs /dev/vg00/rhfs
# mkdir /hfs
# mount /dev/vg00/hfs /hfs
```

2. Copy the lab files to the newly created HFS file system.

```
# cp /home/h4262/disk/lab1/disk_long /hfs
# cp /home/h4262/disk/lab1/make_files /hfs
```

Next, execute the `make_files` program to create five 4-MB ASCII files.

```
# cd /hfs
# ./make_files
```

3. Purge the buffer cache of this data, by unmounting and remounting the file system.

```
# cd /
# umount /hfs
# mount /dev/vg00/hfs /hfs
# cd /hfs
```

Solutions

4. Time how long it takes to read the files with the `cat` command. Record the results below:

```
# timex cat file* > /dev/null
```

real:

user:

sys:

Answer:

```
# timex cat file* > /dev/null (rp2430)
```

```
real      1.04
user      0.01
sys       0.16
```

```
# timex cat file* > /dev/null (rx2600)
```

```
real      0.45
user      0.00
sys       0.05
```

The `cat` command took 1.04 seconds to complete on the `rp2430` and 0.45 seconds on the `rx2600`.

5. In a 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?

Answer:

```
# sar -d 5 200 (rp2430)
```

```
HP-UX r206c41 B.11.11 U 9000/800 03/23/04
```

	device	%busy	avque	r+w/s	blks/s	avwait	avserv
11:53:15							
11:53:20	c1t15d0	5.20	0.50	13	66	5.09	4.54
	c3t15d0	33.60	6922.08	950	15049	629.53	14.85
11:53:25	c1t15d0	7.57	0.50	10	36	5.40	6.82
	c3t15d0	55.98	5215.11	1758	27980	2113.38	13.70
11:53:30	c1t15d0	2.01	0.50	6	44	3.92	5.01
	c3t15d0	100.00	8156.62	2983	47696	2591.43	16.45
11:53:35	c1t15d0	8.00	5.80	18	108	25.31	18.95
	c3t15d0	84.20	1237.19	558	8670	1555.06	17.68

Solutions

```

11:53:40 c1t15d0    6.00    0.50      15      76      4.69      4.72
          c3t15d0    71.20  7379.94    2168    34537  1322.90    14.77
11:53:45 c1t15d0    0.20    0.50       1       5       0.08      8.35
          c3t15d0    25.80  2375.50    950    15206  3478.83    14.42
11:53:50 c3t15d0     9.20    0.50      16     258      5.06      5.21

```

The disk got up to 100% busy.

The average number of requests in the request queue was about 5200.

The average wait time in the request queue was about 1950 ms.

```
# timex ./disk_long
```

```

real      22.76
user       4.57
sys        3.45

```

The operation completed in 22.76 seconds.

```
# sar -d 5 200 (rx2600)
```

```
HP-UX r265c145 B.11.23 U ia64    04/07/04
```

```

13:20:25 device    %busy    avque    r+w/s    blks/s    avwait    avserv
13:20:30 c2t1d0    4.39     0.50     27       706       0.00      1.67
          c2t0d0    27.15     0.50     90       756       0.00      3.04
13:20:35 c2t1d0    41.00    104.29    245      4026     173.18    12.76
          c2t0d0    99.20  24004.63  3322     53129   2127.15    2.35
13:20:40 c2t1d0     1.40     0.50      3        51       0.00      4.62
          c2t0d0   100.00  20020.69  3895     62320   6436.22    2.04
13:20:45 c2t1d0     4.00     0.50     13       287       0.00      5.68
          c2t0d0    57.20  5030.77  2097     33482   9701.92    2.06
13:20:50 c2t1d0     2.40     0.50      7       164       0.00      6.94
13:20:55 c2t1d0     1.40     0.50      2        34       0.00      9.94

```

The disk got up to 100% busy.

The average number of requests in the request queue was about 50,000.

The average wait time in the request queue was about 6100 ms.

```
# timex ./disk_long
```

```

real      16.87
user       0.83
sys        1.96

```

The operation completed in 16.87 seconds.

Solutions

6. Performance tuning — recreate the file system with larger fragment and file system block sizes.

Tuning the size of the fragments and file system blocks can improve performance for sequentially accessed files. The procedure for creating a new file system with customized fragments of 8 KB and file system blocks of 64 KB is shown below:

```
# lvcreate -n custom-lv vg00
# lvextend -L 512 /dev/vg00/custom-lv /dev/dsk/cXtYdZ
# newfs -F hfs -f 8192 -b 65536 /dev/vg00/rcustom-lv
# mkdir /cust-hfs
# mount /dev/vg00/custom_lv /cust-hfs
```

7. Copy the lab files to the customized HFS file system, execute the `make_files` program, and purge the buffer cache.

```
# cp /hfs/disk_long /cust-hfs
# cp /hfs/make_files /cust-hfs
# cd /cust-hfs
# ./make_files
# cd /
# umount /cust-hfs
# mount /dev/vg00/custom-lv /cust-hfs
# cd /cust-hfs
```

8. Time how long it takes to read the files with the `cat` command. Record the results below:

```
# timex cat file* > /dev/null

real:
user:
sys:
```

Answer:

```
# timex cat file* > /dev/null (rp2430)

real      0.84
user      0.01
sys       0.10
```

```
# timex cat file* > /dev/null (rx2600)
```

```
real      0.43
user      0.00
sys       0.03
```

The cat command took 0.84 seconds to complete on the rp2430 and 0.43 seconds on the rx2600.

How do the results of step 8 compare to the default HFS block and fragment results from step 4?

Answer:

The larger block and fragment size resulted in I/O operations which were almost 20% faster on the rp2430 and marginally faster on the rx2600.

9. Performance tuning — change file system mount options.

The manner in which the file system is mounted can impact performance. The **fsasync** mount option can improve performance, but data (metadata) integrity is not as reliable in the event of a crash, and **fsck** could run into difficulties.

```
# cd /

# umount /hfs

# mount -o fsasync /dev/vg00/hfs /hfs

# cd /hfs
```

10. In a 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 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?

Answer:

```
# sar -d 5 200      (rp2430)
```

```
HP-UX r206c41 B.11.11 U 9000/800    03/23/04
```

Solutions

Time	Device	%busy	avque	r+w/s	blks/s	avwait	avserv
12:08:22	device						
12:08:27	c1t15d0	6.20	0.50	9	38	4.18	6.19
	c3t15d0	61.20	5592.30	2120	33818	1376.80	13.94
12:08:32	c1t15d0	7.00	0.50	16	81	4.31	5.28
	c3t15d0	58.60	7186.64	1675	26765	1295.53	17.00
12:08:37	c1t15d0	8.40	3.94	24	146	20.12	13.03
	c3t15d0	92.80	4986.82	1860	29579	2678.62	16.11
12:08:42	c1t15d0	6.60	0.50	17	120	4.84	3.79
	c3t15d0	100.00	15588.44	2344	37493	2943.35	16.95
12:08:47	c3t15d0	71.20	5725.86	2292	36664	6159.69	15.69

The disk got up to 100% busy.

The average number of requests in the request queue was about 7800.

The average wait time in the request queue was about 2900 ms.

```
# timex ./disk_long
```

```
real      17.17
user       4.61
sys        3.72
```

The operation completed in 17.17 seconds.

```
# sar -d 5 200      (rx2600)
```

```
HP-UX r265c145 B.11.23 U ia64      04/07/04
```

Time	Device	%busy	avque	r+w/s	blks/s	avwait	avserv
13:39:39	device						
13:39:44	c2t1d0	1.00	0.50	4	67	0.00	2.51
	c2t0d0	46.11	22190.48	1274	20184	1026.94	2.54
13:39:49	c2t1d0	2.00	0.50	5	77	0.00	5.94
	c2t0d0	100.00	30303.60	3684	58941	4021.91	2.15
13:39:54	c2t1d0	3.20	5.20	9	141	11.85	12.77
	c2t0d0	99.80	11176.41	3888	62008	8740.46	2.05
13:39:59	c2t1d0	0.80	0.50	2	30	0.00	4.42
	c2t0d0	5.60	716.00	287	4562	11067.58	1.51
13:40:04	c2t1d0	4.00	0.50	9	43	0.00	4.45

The disk got up to 100% busy.

The average number of requests in the request queue was about 17500.

The average wait time in the request queue was about 6100 ms.

```
# timex ./disk_long
```

```
real      14.46
user       0.86
sys        3.04
```

The operation completed in 14.46 seconds.

How do the results of step 10 compare to the default mount options in step 5?

Answer:

With **fsasync** turned on, the operation was about 25% faster on the rp2430 and 14% faster on the rx2600.

10-23. LAB: JFS File System Tuning

Directions

The following lab exercise compares performance of JFS with different mount options. The mount options used with JFS can have a big impact on JFS performance.

1. Mount a JFS file system to be used for this lab under `/vxfs`.

```
# mount /dev/vg00/vxfs /vxfs
```

2. Because the above `mount` command specified no special mount options, the default mount options are used. Use the `mount -v` command to view the default options, including the option for transaction logging type.

What type of transaction logging does JFS use by default?

Answer Full logging

3. Change directory to `/vxfs`. Time the execution of the `disk_long` program, which writes 400 MB of data to the file system in 20 MB increments. After each 20 MB is written, the files are deleted. Run the command three times and record the middle results.

```
# cd /vxfs
```

```
# timex ./disk_long
# timex ./disk_long
# timex ./disk_long
```

Record middle results:

Real: _____ User: _____ Sys: _____

Answer Varies with configuration, live data from test

```
# timex ./disk_long (rp2430)
```

```
real      12.34
user       4.82
sys        3.45
```

```
# timex ./disk_long(rrx2600)
```

```
real       9.49
user       0.90
sys        1.62
```

If you look back to the HFS results, you will see that this is faster. See question 5 from the previous lab; test time was 23 seconds or 17 seconds!

4. Remount the JFS file system using **delaylog** option. This helps performance of non-critical transactions. Run the command three times and record the middle results.

```
# cd /
# umount /vxfs
# mount -o delaylog /dev/vg00/vxfs /vxfs
# cd /vxfs
# timex ./disk_long
# timex ./disk_long
# timex ./disk_long
```

Record middle results:

Real: _____ User: _____ Sys: _____

Answer Varies with configuration, should be faster than before:

```
# timex ./disk_long      (rp2430)
```

```
real      10.93
user       4.85
sys        3.52
```

```
# timex ./disk_long      (rx2600)
```

```
real      9.23
user       0.90
sys        1.64
```

Based on the results, does the **disk_long** program perform any non-critical transactions?

Answer

The answer is *yes*; the **disk_long** program is performing some non-critical transactions. This is seen by some improvement in time to execute. Since the programs write data in 1 MB increments (that's it), just about every JFS transaction is critical, so mounting with **delaylog** versus **full log** does not greatly affect performance in this case. It will in other cases.

5. Remount the JFS file system using **tmplog** option. This causes the system call to be returned *after* the JFS transaction is updated in memory (step 1 from lecture), and *before* the transaction is written to the intent log. Run the command three times and record the middle results.

```
# cd /
# umount /vxfs
```

Solutions

```
# mount -o tmplog /dev/vg00/vxfs /vxfs
# cd /vxfs
# timex ./disk_long
# timex ./disk_long
# timex ./disk_long
```

Record middle results:

Real: _____ User: _____ Sys: _____

Answer Varies with configuration, live test data:

```
# timex ./disk_long (rp2430)
```

```
real      10.08
user       4.82
sys        3.40
```

```
# timex ./disk_long (rx2600)
```

```
real      10.35
user       0.90
sys        1.60
```

Based on the results, why does the `disk_long` program show little or no improvement when mounted with `tmplog`?

Answer

The `disk_long` program shows little performance improvement because the program is performing **extending write** calls. When an “extending write” call is issued, by default JFS writes the user data *first* before writing the JFS transaction to the intent log. As a result, even JFS file systems mounted with `tmplog` or `nolog` will still have to wait for the user data to be written to disk. This waiting for the user data to be written hurts the performance of JFS.

6. Remount the JFS file system using `tmpcache` option. This allows the JFS transaction to be created without having to wait for the user data to be written in extending write calls. Run the command three times and record the middle results.

```
# cd /
# umount /vxfs
# mount -o mincache=tmpcache /dev/vg00/vxfs /vxfs
# cd /vxfs
# timex ./disk_long
# timex ./disk_long
# timex ./disk_long
```

Record middle results:

Real: _____ User: _____ Sys: _____

Answer Varies with configuration, live test data. Fastest yet!

```
# timex ./disk_long (rp2430)
```

```
real      9.13
user      4.51
sys       2.69
```

```
# timex ./disk_long (rx2600)
```

```
real      9.51
user      0.90
sys       1.65
```

Answer

When the `mincache=tmpcache` option is specified, under 2 MB out of 400 MB is physically written to disk. When this option is not specified, all 400 MB (400 out of 400) is physically written to disk. Major performance improvements should be seen with using this option, especially for applications doing lots of “extending write” calls (like the one in the lab).

7. Remount the JFS file system using **direct** option. This option requires all user data and all JFS transactions to bypass the buffer cache and go directly to disk. Run the command just once and record the results.

```
# cd /
# umount /vxfs
# mount -o mincache=direct /dev/vg00/vxfs /vxfs
# cd /vxfs
# timex ./disk_long
```

Record results:

Real: _____ User: _____ Sys: _____

Answer Varies with configuration, live test data, not very impressive!

```
# timex ./disk_long (rp2430)
```

```
real      7:36.75
user      5.15
sys       5.41
```

```
# timex ./disk_long (rx2600)
```


Solutions

```
real      3:06.72
user       0.90
sys       2.45
```

Based on the results, why does the `disk_long` program show such poor performance results when mounted with `mincache=direct`? When would this option be appropriate to use?

Answer

The performance is poor because system calls have to wait while user data *and* JFS transactions are written out to disk. Normally, the JFS transactions are written to buffer cache, and the system calls do not have to wait for the transaction to be written to disk.

This option is appropriate when the application performs its own caching, like with an RDBMS (for example, Oracle).

8. Dismount the VxFS file system.

```
# umount /vxfs
```

11–20. LAB: Network Performance

Directions

The following two labs investigate network read and write performance. The labs use NFS and are performed against the JFS file system created in the JFS module.

Lab 1 — Network Read Performance

To perform this lab, two systems are needed: an NFS server and an NFS client. Pair up with another student in the class for this lab.

1. Make sure the JFS file system on the server contains the `make_files` program. Execute the `make_files` program to create files for the client to access.

```
# mount /dev/vg00/vxfs /vxfs
# cp /home/h4262/disk/lab1/make_files /vxfs
# cd /vxfs
# ./make_files
```

2. Export the JFS file system so the client can mount it.

```
# exportfs -i -o root=<client_hostname> /vxfs
# exportfs
```

3. From the client system, mount the NFS file system.

```
# umount /vxfs
# mount server_hostname:/vxfs /vxfs
```

4. Time how long it takes to read the 20 MB of files from the mounted file system. Record the results:

```
# timex cat /vxfs/file* > /dev/null
```

Record results:

Real: _____ User: _____ Sys: _____

Answer Varies with configuration, live test data below,

```
# timex cat /vxfs/file* > /dev/null (rp2430)
```

```
real      1.80
user      0.01
sys       0.07
```

```
# timex cat /vxfs/file* > /dev/null (rx2600)
```

```
real      1.17
user      0.00
```

Solutions

```
sys          0.02
```

5. Now that the data is in the client's buffer cache, time how long it takes to read the exact same files again. Record the results:

```
# timex cat /vxfs/file* > /dev/null
```

Record results:

Real: _____ User: _____ Sys: _____

Answer Varies with configuration, live data below. Much faster once buffered.

```
# timex cat /vxfs/file* > /dev/null (rp2430)
```

```
real      0.05
user      0.01
sys       0.04
```

```
# timex cat /vxfs/file* > /dev/null (rx2600)
```

```
real      0.02
user      0.00
sys       0.01
```

Moral: Try to have a big enough buffer on the client system for a lot of data to be cached. Also, **biod** daemons will help by prefetching data.

6. Test to see if fewer **biod** daemons will change the initial performance.

```
# cd /
# umount /vxfs
# kill $(ps -e | grep biod | cut -c1-7)
# /usr/sbin/biod 4
# mount server_hostname:/vxfs /vxfs
# timex cat /vxfs/file* > /dev/null
```

Record results:

Real: _____ User: _____ Sys: _____

Answer Varies with configuration, but significant change. Large sequential access appears to be independent of the number of **biods**. Not what theory suggests? Well this depends!

```
# timex cat /vxfs/file* > /dev/null (rp2430)
```

```
real      1.80
user      0.01
sys       0.07
```

```
# timex cat /vxfs/file* > /dev/null (rx2600)
```

```

real          1.15
user          0.00
sys           0.02

```

- Once finished, remove the files and **umount** the file system.

```

# rm /vxfs/file*
# umount /vxfs

```

Lab 2 — Network Write Performance

The following lab has the client perform many writes to an NFS file system. The following parameters will be investigated:

- Number of **biod** daemons
- NFS version 2 versus NFS version 3
- TCP versus UDP

During this lab, the monitoring tools shown below should be used on the client and server

CLIENT

```

# nfsstat -c
# glance NFS report (n key)
# glance Global Process (g key)
    - monitor biod daemons
# glance Disk report (d key)
    - monitor Remote Rds/Wrts

```

SERVER

```

# nfsstat -s
# glance NFS report (n key)
# glance Global Process (g key)
    -monitor nfsd daemons

```

- From the NFS client, mount the NFS file system as a version 2 file system.

```
# mount -o vers=2 server_hostname:/vxfs /vxfs
```

- Terminate all the **biod** daemons on the client.

```
# kill $(ps -e |grep biod|cut -c1-7)
```

- Time how long it takes to copy the **vmunix** file to the mounted NFS file system. Record the results: The first command buffers the file.

```

# cat /stand/vmunix >/dev/null
# timex cp /stand/vmunix /jfs

```

Record results:

Real: _____ User: _____ Sys: _____

Solutions

Answer Varies with configuration

```
# timex cp /stand/vmunix /vxfs      (rp2430)
```

```
real      33.95
user       0.00
sys        0.44
```

```
# timex cp /stand/vmunix /vxfs      (rx2600)
```

```
real      20.64
user       0.00
sys        0.38
```

4. Now, start up the **biod** daemons, and retry timing the copy. Record the results:

```
# /usr/sbin/biod 4
# timex cp /stand/vmunix /jfs
```

Record results:

Real: _____ User: _____ Sys: _____

Answer Varies with configuration, the test data shows marked improvement. The **biods** are providing the "write behind" service which is reducing the wait time experienced by the **cp** command.

```
# timex cp /stand/vmunix /vxfs      (rp2430)
```

```
real      29.27
user       0.00
sys        0.16
```

```
# timex cp /stand/vmunix /vxfs      (rx2600)
```

```
real      16.53
user       0.00
sys        0.14
```

5. Change the mount options to version 3 and retime the transfer:

```
# cd /
# umount /vxfs
# mount -o vers=3 server_hostname:/vxfs /vxfs
# cd /
# timex cp /stand/vmunix /vxfs
```

Record results:

Real: _____ User: _____ Sys: _____

Answer: Interesting, it would appear that Version 3 mounting is far better than version 2. The results were obtained using the same 4 biods started in question 3.

```
# timex cp /stand/vmunix /vxfs      (rp2430)

real      2.63
user      0.00
sys       0.18
# timex cp /stand/vmunix /vxfs      (rx2600)

real      4.13
user      0.00
sys       0.13
```

6. Compare the speed of FTP to NFS. Transfer the file to the server using the **ftp** utility.

```
# ftp server_hostname
# put /stand/vmunix /vxfs/vmunix.ftp
```

How long did the FTP transfer take? _____

Explain the difference in performance.

Answer The data below shows that **ftp** is well optimized to perform data transfer. The good news is that Version 3 of NFS keeps up with it and remember that at **11i**, NFS is using TCP/IP and not UDP/IP.

```
# ftp r265c69      (rp2430)
Connected to r265c69.cup.edunet.hp.com.
220 r265c69.cup.edunet.hp.com FTP server (Version 1.1.214.4 (PHNE_23950) Tue May
22 05:49:01 GMT 2001) ready.
Name (r265c69:root):
331 Password required for root.
Password:
230 User root logged in.
Remote system type is UNIX.
Using binary mode to transfer files.
ftp> put /stand/vmunix /vxfs/vmunix.ftp
200 PORT command successful.
150 Opening BINARY mode data connection for /vxfs/vmunix.ftp.
226 Transfer complete.
27573440 bytes sent in 2.55 seconds (10554.31 Kbytes/s)
ftp>

# ftp r265c145      (rx2600)
Connected to r265c145.
220 r265c145.cup.edunet.hp.com FTP server (Revision 1.1 Version wuftp-2.6.1 Tue
Jul 15 07:42:07 GMT 2003) ready.
```

Solutions

```
Name (r265c145:root):
331 Password required for root.
Password:
230 User root logged in.
Remote system type is UNIX.
Using binary mode to transfer files.
ftp> put /stand/vmunix /vxfs/vmunix.ftp
200 PORT command successful.
150 Opening BINARY mode data connection for /vxfs/vmunix.ftp.
226 Transfer complete.
47716848 bytes sent in 4.03 seconds (11557.24 Kbytes/s)
ftp>
```

7. Test the potential performance benefit of turning off the new TCP feature of HPUX 11i. First, mount the file system with UDP protocol rather than the default TCP.

```
# umount /vxfs
# mount -o vers=3 -o proto=udp server_hostname:/vxfs /vxfs
```

Perform the copy test again and compare the results with the TCP version 3 mount data in part 3. Is UDP quicker than TCP?

```
# timex cp /stand/vmunix /vxfs
```

Answer

```
# timex cp /stand/vmunix /vxfs (rp2430)
```

```
real    2.44
user    0.00
sys     0.15
```

```
# timex cp /stand/vmunix /vxfs (rx2600)
```

```
real    4.08
user    0.00
sys     0.13
```

It would appear that UDP is marginally quicker than TCP but the difference is very small and probably not worth the risk. HPUX 11i version 3 NFS with TCP provides good performance and reliability.