

Overview of AIX page replacement

Skill Level: Intermediate

[David Hepkin \(dhepkin@us.ibm.com\)](mailto:dhepkin@us.ibm.com)

AIX Kernel Architect

IBM

08 Jan 2008

Go through detailed information on how the AIX® virtual memory manager (AIX VMM) works and how to use tunable parameters to adjust the operation of the AIX VMM. The AIX VMM is responsible for managing all of the memory on a system. The operation of the AIX VMM is critical to the performance of a system, and it also provides several tunable parameters that you can use to optimize its operation for different workloads.

Introduction

The AIX® virtual memory manager (AIX VMM) is a page-based virtual memory manager. A page is a fixed-size block of data. A page might be resident in memory (that is, mapped into a location in physical memory), or a page might be resident on a disk (that is, paged out of physical memory into paging space or a file system).

One unique aspect of the AIX VMM is the management of cached file data. The AIX VMM integrates cached file data with the management of other types of virtual memory (for example, process data, process stack, and so forth). It caches the file data as pages, just like virtual memory for processes.

AIX maps pages into real memory based on demand. When an application references a page that is not mapped into real memory, the system generates a page fault. To resolve the page fault, the AIX kernel loads the referenced page to a location in real memory. If the referenced page is a new page (that is, a page in a data heap of the process that has never been previously referenced), "loading" the referenced page simply means filling a real memory location with zeros (that is, providing a zero-filled page). If the referenced page is a pre-existing page (that is, a page in a file or a previously paged out page), loading the referenced page involves

reading the page from the disk (paging space or disk file system) into a location in real memory.

Once a page is loaded into real memory, it is marked as unmodified. If a process or the kernel modifies the page, the state of the page changes to modified. This allows AIX to keep track of whether a page has been modified after it was loaded into memory.

As the system adds more pages into real memory, the number of empty locations in real memory that can contain pages decreases. You can also refer to the number of empty locations as the number of free page frames. When the number of free page frames gets to a low value, the AIX kernel must empty out some locations in real memory for reuse of new pages. This process is otherwise known as page replacement.

The AIX VMM has background daemons responsible for doing page replacement. A page replacement daemon is referred to as `lrud` (shows up as `lrud` in the output of `ps -k`). `lrud` daemons are responsible for scanning in memory pages and evicting pages in order to empty locations in real memory. When a page replacement daemon determines that it wants to evict a specific page, the page replacement daemon does one of two things:

- If the page is modified, the page replacement daemon writes the page out to a secondary storage location (for example, paging space or file system disk). The physical memory block that contains the page is marked as free and ready for reuse for additional pages.
- If the page is unmodified, the page replacement daemon can simply mark the physical memory block as free, and the physical memory block can then be re-used for another page. In this case, the page replacement daemon does not have to write the page out to disk, because the in-memory version of the page is unmodified, and thus is identical to the copy of the page that resides on the disk (in paging space or on a disk file system).

The page replacement daemons target different types of pages for eviction based on system memory usage and tunable parameters. The remainder of this article provides details on how the page replacement daemons target pages for eviction.

Page types

Fundamentally, there are two types of pages on AIX:

- Working storage pages
- Permanent storage pages

Working storage

Working storage pages are pages that contain *volatile* data (in other words, data that is not preserved across a reboot). On other platforms, working storage memory is sometimes referred to as *anonymous* memory. Examples of virtual memory regions that consist of working storage pages are:

- Process data
- Stack
- Shared memory
- Kernel data

When modified working storage pages need to be paged out (moved from memory to the disk), they are written to paging space. Working storage pages are never written to a file system.

When a process exits, the system releases all of its private working storage pages. Thus, the system releases the working storage pages for the data of a process and stack when the process exits. The working storage pages for shared memory regions are not released until the shared memory region is deleted.

Permanent storage

Permanent storage pages are pages that contain permanent data (that is, data that is preserved across a reboot). This permanent data is just file data. So, permanent storage pages are basically just pieces of files cached in memory.

When a modified permanent storage page needs to be paged out (moved from memory to disk), it is written to a file system. As mentioned earlier, an unmodified permanent storage page can just be released without being written to the file system, since the file system contains a pristine copy of the data.

For example, if an application is reading a file, the file data is cached in memory in permanent storage pages. These permanent storage pages are unmodified, meaning that the pages have not been modified in memory. So, the in-memory permanent storage pages are equivalent to the file data on the disk. When AIX needs to free up memory, it can just "release" these pages without having to write anything to disk. If the application had been doing writes to the file instead of reads, the permanent storage pages would be "modified," and AIX would have to flush the pages to disk before releasing the pages.

You can divide permanent storage pages into two sub-types:

- Client pages

- Non-client pages

Non-client pages are pages containing cached Journaled File System (JFS) file data. Non-client pages are sometimes referred to as persistent pages. Client pages are pages containing cached data for all other file systems (for example, JFS2 and Network File System (NFS)).

Page classification

In order to help optimize which pages are selected for replacement by the page replacement daemons, AIX classifies pages into one of two types:

- Computational pages
- Non-computational pages

Computational pages are pages used for the text, data, stack, and shared memory of a process. Non-computational pages are pages containing file data for files that are being read and written.

How pages get classified

All working storage pages are computational. A working storage page is never marked as non-computational.

Depending on how you use the permanent storage pages, the pages can be computational or non-computational. If a file contains executable text for a process, the system treats the file as computational and marks all of the permanent storage pages in the file as computational. If the file does not contain executable text, the system treats the file as non-computational file and marks all of the pages in the file as non-computational.

When you first open a file, the AIX kernel creates an internal VMM object to represent the file. It marks it as non-computational, meaning all files start out as non-computational.

As a program does reads and writes to the file, the AIX kernel caches the file's data in memory as non-computational permanent storage pages.

If the file is closed, the AIX kernel continues to cache the file data in memory (in permanent storage pages). The kernel continues to cache the file for performance; for example, if another process comes along later and uses the same file, the file data is still in memory, and the AIX kernel does not have to read the file data in from disk.

When a page fault is taken on a file due to an instruction fetch, only the non-computational file transitions to the computational state. When a process page faults on a file (meaning the process references a part of the file that is not currently cached in memory in a permanent storage page), the process generates a page fault. If the page fault is due to an instruction fetch (meaning the process was trying to load an instruction from the page to execute), the kernel marks the file as computational. This involves marking all pages in the file as computational. A file is either completely computational or non-computational.

Once a file has been marked as computational, it remains marked as a computational file until the file is deleted (or the system is rebooted). Thus, a file remains marked as computational even after it is moved or renamed.

Page replacement

The AIX page replacement daemons scan memory a page at a time to find pages to evict in order to free up memory. The page replacement daemons must choose pages carefully to minimize the performance impact of paging on the system, and the page replacement daemons target pages of different classes based on tunable parameter settings and system conditions.

There are a number of tunable parameters that you can use to control how AIX selects pages to replace.

minperm and maxperm

The two most basic page replacement tunable parameters are `minperm` and `maxperm`. These tunable parameters are used to indicate how much memory the AIX kernel should use to cache non-computational pages. The `maxperm` tunable parameter indicates the maximum amount of memory that should be used to cache non-computational pages.

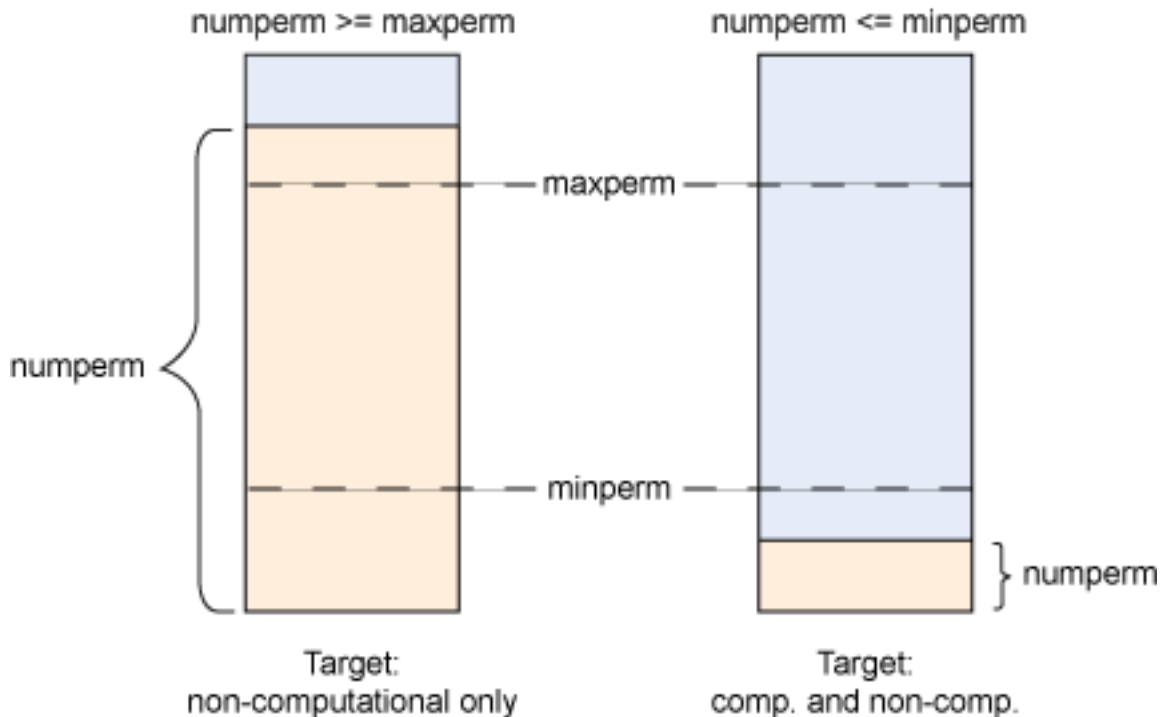
By default, `maxperm` is an "un-strict" limit, meaning that the limit can be exceeded. Making `maxperm` an un-strict limit allows more non-computational files to be cached in memory when there is available free memory. The `maxperm` limit can be made a "strict" limit by setting the `strict_maxperm` tunable parameter to 1. When `maxperm` is a strict-limit, the kernel does not allow the number of non-computational pages to exceed `maxperm`, even if there is free memory available. Thus, the disadvantage with making `maxperm` a strict limit is that the number of non-computational pages cannot grow beyond `maxperm` and consume more memory when there is free memory on the system.

The `minperm` limit indicates the target minimum amount of memory that should be used for non-computational pages.

The number of non-computational pages is referred to as `numperm`: The `vmstat -v` command displays the `numperm` value for a system as a percentage of a system's real memory.

Figure 1 below gives an overview of how these tunable parameters work under different system conditions:

Figure 1. minperm and maxperm limits



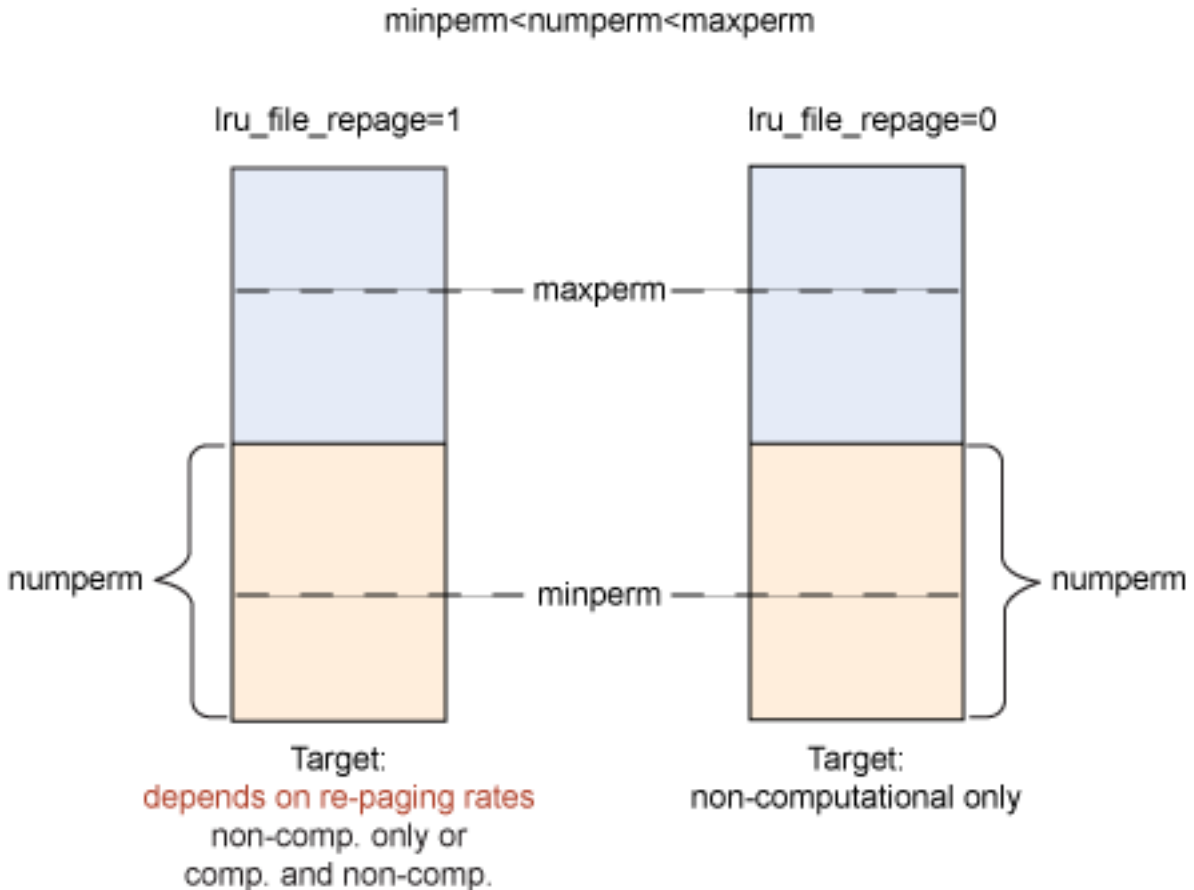
When the number of non-computational pages (`numperm`) is greater than or equal to `maxperm`, the AIX page replacement daemons strictly target non-computational pages (for example, cached files that are not executables).

When the number of non-computational pages (`numperm`) is less than or equal to `minperm`, the AIX page replacement daemons target both computational and non-computational pages. In this case, AIX scans both classes of pages and evicts the least recently used pages.

When the number of non-computational pages (`numperm`) is between `minperm` and `maxperm`, the `lru_file_repage` tunable parameter controls what kind of pages the AIX page replacement daemons should steal (see Figure 2).

lru_file_repage

Figure 2. lru_file_repage tunable parameter



When `numperm` is between `minperm` and `maxperm`, the AIX page replacement daemons determine what type of pages to target based on their internal re-paging table when the `lru_file_repage` tunable parameter is set to 1.

The AIX kernel maintains a re-paging table in order to identify pages that are paged out and then quickly paged back in. When the kernel pages a page out and then pages it back in, it usually indicates that there is strong demand for the page and that the page should stay in memory. The kernel maintains an indication of how many times it re-pages computational pages and how many times it re-pages non-computational pages. The AIX kernel can then use this information to determine which class of pages is being re-paged more heavily (thus, indicating which class of pages is experiencing higher demand). When the `lru_file_repage` tunable parameter is set to 1, the AIX kernel uses this re-paging information to determine whether to target just non-computational pages or target both computational and non-computational pages. If the rate of re-paging computational pages is higher than that of non-computational pages, the AIX kernel just targets non-computational pages (since there appears to be stronger demand for computational pages). If the rate of re-paging non-computational pages is higher than that of computational pages, the AIX kernel targets both computational as well as non-computational pages.

In most customer environments, it is most optimal to just have the kernel always target non-computational pages, because paging computational pages (for example, a process's stack, data, and so forth) usually has a much higher performance cost on a process than paging non-computational pages (that is, data file cache). Thus, the `lru_file_repage` tunable parameter can be set to 0. In this case, the AIX kernel always targets non-computational pages when `numperm` is between `minperm` and `maxperm`.

maxclient

In addition to the `minperm` and `maxperm` tunable parameters, there is also a `maxclient` tunable parameter. The `maxclient` tunable parameter specifies a limit on the maximum amount of memory that should be used to cache non-computational client pages. Because all non-computational client pages are a subset of the total number of non-computational permanent storage pages, the `maxclient` limit must always be less than or equal to the `maxperm` limit.

The number of non-computational client pages is referred to as `numclient`. The `vmstat -v` command displays the `numclient` value for a system as a percentage of a system's real memory.

By default, the `maxclient` limit is a strict limit. This means that the AIX kernel does not allow the non-computational client file cache to exceed the `maxclient` limit (that is, the AIX kernel does not allow `numclient` to exceed `maxclient`). When `numclient` reaches the `maxclient` limit, the AIX kernel starts page replacement in a special, client-only mode. In this client-only mode, the AIX page replacement daemons strictly target client pages.

Monitoring a system's memory usage

AIX provides several tools for providing information about counts of the different pages on the system.

vmstat command

The `vmstat` command reports information about a system's memory usage and statistics about VMM operations like page replacement.

The `-v` option specified with the `vmstat` command displays the percentage of real memory being used for different classification of pages (see [Listing 1](#)):

Listing 1. vmstat -v command

```
# vmstat -v
4980736 memory pages
```

```

739175 lruable pages
432957 free pages
  1 memory pools
84650 pinned pages
 80.0 maxpin percentage
 20.0 minperm percentage <<- system's minperm% setting
 80.0 maxperm percentage <<- system's maxperm%
setting
  2.2 numperm percentage << % of memory containing
non-comp. pages
16529 file pages <<- # of non-comp. pages
  0.0 compressed percentage
  0 compressed pages
  2.2 numclient percentage <<- % of memory containing
non-comp. client pages
 80.0 maxclient percentage <<- system's maxclient%
setting
16503 client pages <<- # of client pages
  0 remote pageouts scheduled
  0 pending disk I/Os blocked with no pbuf
  0 paging space I/Os blocked with no psbuf
2484 filesystem I/Os blocked with no fsbuf
  0 client filesystem I/Os blocked with no fsbuf
  0 external pager filesystem I/Os blocked with no fsbuf
  0 Virtualized Partition Memory Page Faults
 0.00 Time resolving virtualized partition memory page faults

```

So, in the above example, there are 16529 non-computational file pages mapped into memory. These non-computational pages consume 2.2 percent of memory. Of these 16529 non-computational file pages, 16503 of them are client pages.

The `vmstat` output does not provide information about computational file pages. Information about computational file pages can be gathered from the `svmon` command.

svmon command

Another command that can be used to display information about a system's memory usage is the `svmon` command. The `svmon` command supports a number of different options for providing very detailed information about a system's memory usage.

The `-G` option to the `svmon` command displays information about how much memory is being used for different types of pages (see [Listing 2](#)):

Listing 2. -G option to svmon command

```

# svmon -G
      size      inuse      free      pin      virtual
memory  786432    209710    576722    133537    188426
pg space  131072      1121
      work      pers      clnt
pin     133537      0        0
in use  188426      0      21284
PageSize PoolSize      inuse      pgsp      pin      virtual

```

s	4 KB	-	103966	1121	68929	82682
m	64 KB	-	6609	0	4038	6609

To understand how a system's real memory is being used, `svmon` displays three columns:

- `work` —working storage
- `pers` —persistent storage (Persistent storage pages are non-client pages—that is, JFS pages.)
- `clnt` —client storage

For each page type, `svmon` displays two rows:

- `inuse` —number of 4K pages mapped into memory
- `pin` —number of 4K pages mapped into memory and pinned (pin is a subset of `inuse`)

So, in the above example, there are 188426 working storage pages mapped into memory. Of those 188426 working storage pages, 133537 of them are pinned (that is, can't be paged out).

There are no persistent storage pages (because there are no JFS filesystems in use on the system). There are 21284 client storage pages, and none of them are pinned.

The `svmon` command does not display the number of permanent storage pages, but it can be calculated from the `svmon` output. As mentioned earlier, the number of permanent storage pages is the sum of the number of persistent storage pages and the number of client storage pages. So, in the above example, there are a total of 21284 permanent storage pages on the system:

```
0 persistent storage pages + 21284 client storage pages = 21284 permanent storage pages
```

The type of information reported by `svmon` is slightly different than `vmstat`. `svmon` reports information about the number of in-memory pages of different types—working, persistent (that is, non-client), and client. `svmon` does not report information about computational versus non-computational. `svmon` just reports the total number of in-memory pages of each page type.

In contrast, `vmstat` reports information about non-computational versus computational pages.

To illustrate this difference, consider the above example of `svmon` output. Some of

the 21284 client pages will be computational, and the rest of the 21284 client pages will be non-computational. To determine the breakdown of these client pages between computational and non-computational, use the `vmstat` command to determine how many of the 21284 client pages are non-computational.

Displaying and setting tunable parameters

The `vm0` command is used to interact with VMM tunable parameters. The `vm0` command can be used to display information about tunable parameters as well as to set the values for tunable parameters.

To display the current values of all VMM tunable parameters, run the `vm0` command with the `-L` option:

```
# vm0 -L
```

To display the current values of select VMM tunable parameters, use the `-L` option to list names of tunable parameters. For example, the following command snapshot shows output when listing the current values of the `minperm%`, `maxperm%`, `maxclient%`, and `lru_file_repage` tunable parameters (see [Listing 3](#)):

Listing 3. Tunable parameters

```
# vm0 -L minperm% -L maxperm% -L maxclient% -L lru_file_repage
NAME                                CUR    DEF    BOOT  MIN    MAX    UNIT          TYPE
DEPENDENCIES
-----
lru_file_repage                    1      1      1     0      1     boolean       D
-----
maxclient%                          80     80     80     1     100    % memory      D
  maxperm%
  minperm%
-----
maxperm%                             80     80     80     1     100    % memory      D
  minperm%
  maxclient%
-----
minperm%                             20     20     20     1     100    % memory      D
  maxperm%
  maxclient%
```

Table 1. Tunable parameters: Column descriptions

Column	Description
CUR	This column lists the current values of the tunable parameters.
DEF	This column lists the default values.
BOOT	This column lists the values of the tunable

	parameters at the time the system was booted.
MIN	This column lists the minimum values of the tunable parameters.
MAX	This column lists the maximum values of the tunable parameters.
UNIT	This column lists the unit in which the tunable parameter is specified.

The `vmo` command supports changing the value of a tunable parameter immediately or to defer changing the value of a tunable parameter until the system is rebooted. To change the above tunable parameters and have the changes take effect immediately and on all subsequent reboots, specify the `-p` option. Here is an example (see [Listing 4](#)):

Listing 4. -p option

```
# vmo -p -o lru_file_repage=0 -o maxclient%=90 -o maxperm%=90 -o minperm%=3
Setting minperm% to 3 in nextboot file
Setting maxperm% to 90 in nextboot file
Setting maxclient% to 90 in nextboot file
Setting lru_file_repage to 0 in nextboot file
Setting minperm% to 3
Setting maxperm% to 90
Setting maxclient% to 90
Setting lru_file_repage to 0
```

Suggested tunable parameter settings

The vast majority of workloads benefit from the VMM page replacement daemons targeting non-computational pages. Thus, the following suggested tunable parameters provide the best performance for the majority of workloads (see [Listing 5](#)):

Listing 5. Tunable parameters with best performance

```
lru_file_repage = 0
maxperm = 90%
maxclient = 90%
minperm = 3%
strict_maxclient = 1 (default)
strict_maxperm = 0 (default)
```

These tunable parameters can be set with the `vmo` command (see [Listing 6](#)):

Listing 6. Tunable parameters set with the vmo command

```
# vmo -p -o lru_file_repage=0 -o maxclient%=90 -o maxperm%=90 -o minperm%=3
# vmo -p -o strict_maxclient=1 -o strict_maxclient=0
```

The settings can be viewed with the `vmo -L` command.

These tunable parameter settings apply to AIX Version 5.2 and AIX Version 5.3. To set these tunable parameters on AIX Version 5.2, AIX Version 5.2 TL6 or later is required. To set the above tunable parameters on AIX Version 5.3, AIX Version 5.3 TL1 or later is required.

The above tunable parameters settings are the default settings for AIX Version 6.1.

Conclusion

The AIX VMM classifies pages based on use. You can use system tunable parameters to control the behavior of the AIX page replacement daemons and control how AIX treats different classes of pages page replacement. Tuning the AIX VMM can result in significant performance improvements for workloads.

Share this...



Digg
this
story



Post
to
del.icio.us



Slashdot
it!

Resources

Learn

- See [VMM page replacement tuning](#) for more information.
- [IBM Redbooks](#): The AIX 5L Practical Performance Tools and Tuning Guide is a comprehensive guide about AIX performance monitoring and tuning tools.
- [Popular content](#): See what AIX and UNIX® content your peers find interesting.
- [AIX and UNIX](#): The AIX and UNIX developerWorks zone provides a wealth of information relating to all aspects of AIX systems administration and expanding your UNIX skills.
- [New to AIX and UNIX?](#): Visit the "New to AIX and UNIX" page to learn more about AIX and UNIX.
- [AIX Wiki](#): A collaborative environment for technical information related to AIX.
- Search the AIX and UNIX library by topic:
 - [System administration](#)
 - [Application development](#)
 - [Performance](#)
 - [Porting](#)
 - [Security](#)
 - [Tips](#)
 - [Tools and utilities](#)
 - [Java™ technology](#)
 - [Linux](#)
 - [Open source](#)
- [Safari bookstore](#): Visit this e-reference library to find specific technical resources.
- [developerWorks technical events and webcasts](#): Stay current with developerWorks technical events and webcasts.
- [Podcasts](#): Tune in and catch up with IBM technical experts.

Get products and technologies

- [IBM trial software](#): Build your next development project with software for

download directly from developerWorks.

Discuss

- Participate in the [developerWorks blogs](#) and get involved in the developerWorks community.
- Participate in the AIX and UNIX forums:
 - [AIX —technical forum](#)
 - [AIX 6 Open Beta](#)
 - [AIX for Developers Forum](#)
 - [Cluster Systems Management](#)
 - [IBM Support Assistant](#)
 - [Performance Tools—technical](#)
 - [Virtualization—technical](#)
 - [More AIX and UNIX forums](#)

About the author

David Hepkin

David is an AIX kernel architect. His responsibilities include designing and developing new technology for the AIX operating system. His background is in kernel development. You can contact him at dhepkin@us.ibm.com.

Trademarks

IBM and AIX are registered trademarks of International Business Machines Corporation in the United States, other countries, or both.

Java and all Java-based trademarks are trademarks of Sun Microsystems, Inc. in the United States, other countries, or both.

Linux is a trademark of Linus Torvalds in the United States, other countries, or both.

UNIX is a registered trademark of The Open Group in the United States and other countries.