Chapter 5

Configuring an HP-UX Kernel

Introduction

Kernel management in HP-UX 11i Version 2 is done with a set of kernel configuration commands or through the web-based **kcweb** tool. This chapter covers the kernel-related commands, gives some examples of using the commands to modify and build kernels, and gives an overview of **kcweb**.

Most applications require that you to modify your kernel in some way. You may need to create a new HP-UX kernel to add device drivers or subsystems, tune the kernel to get improved performance, alter configurable parameters, or change the dump and swap devices. If you update or modify a dynamic element of your kernel, as shown in an example in this chapter, a reboot is not required. Updating or modifying a static element requires a reboot and may also require some additional steps, which a later example shows.

This chapter covers the following topics:

- Overview, examples, and running kernel-related commands: **kcmodule**, **kctune**, **kconfig**, **kclog**, **kcalarm**, **kcusage**, and **kcmod**
- A flowchart showing a typical kernel rebuild
- Example of using the entries in the flowchart to rebuild a kernel
- Example of booting a saved kernel configuration

- The **system** file and making multiple kernel changes using it
- Overview of the Web-based kernel tool **kcweb**

Kernel Commands

A new set of kernel-related commands has been developed that have a common behavior. This section covers all the commands and provides examples of using some of them.

kcmodule

kcmodule queries and changes kernel modules in the currently running kernel configuration or a saved kernel configuration that you are staging for future use. Hundreds of modules are present in an HP-UX kernel that consists of device drivers, kernel subsystems, and other kernel code.

kcmodule with no options provides the modules in your system and both their current state and the state on next boot if any changes are pending, as shown in this abbreviated example:

```
# kcmodule
Module           State   Cause     Notes
DeviceDriver     unused
KeyboardMUX      unused
LCentIf          static  best
MouseMUX         unused
UsbBootKeyboard  unused
UsbBootMouse     unused
UsbHub           unused
UsbMiniBus       unused
UsbOhci          unused
acpi_node        static  best
arp              static  depend
asio0            static  best
audio            static  best
autofsc          static  best
azusa_psm        static  best
beep             static  depend
btlan            static  best
c460gx_psm       static  depend
c8xx             static  best
cachefsc         static  best
ccio             unused
cdfs             auto    best      auto-loadable, unloadable
cec_hp           static  depend
cell             static  best
cifs             static  best
clone            static  best
consp1           unused
```

```
diag2            static  best
dlpi             static  best
dm_sample_fsid   unused
dmapi            unused
dmp              static  depend
dmsample         unused
echo             static  best
ehci             unused
fcd              static  best
fcms             static  depend
fcp              static  depend
fcp_cdio         static  depend
fcparray         static  depend
fcpdev           static  depend
fcpmux           static  depend
fddi4            static  best
ffs              static  best
framebuf         unused
gelan            static  best
graph3           unused
gvid             unused
gvid_core        unused
hcd              unused
hid              unused
hpstreams        static  best
hsx              static  explicit  loadable, unloadable
hub              unused
ia64_psm         static  best
idds             unused
                   .
                   .
                   .
```

This abbreviated output shows some of the modules in the system without much detail. We modify the last module shown, idds, in a later example. To get detailed information on a specific module, use the -v option, as shown for vxfs:

```
# kcmodule -v vxfs
Module              vxfs  [3F559170]
Description         Veritas Journal File System (VxFS)
State               static (best state)
State at Next Boot  static (best state)
Capable             static unused
Depends On          module libvxfs:0.0.0
                    interface HPUX_11_23:1.0

#
```

This verbose output shows more information for the specific module that we specified, or would have shown verbose output for every module if I had not specified the name of a specific module.

For every module in the verbose output, there is a name and description such as a name of vxfs, the version number in square brackets after the name, and a short description of the module in the example. It is possible for multiple versions to be listed if, for instance, the currently running kernel uses a different version than will be used on the next boot.

The state of the module is relative to the currently running kernel (which is shown in the example), the next boot (both the currently running and next boot states are shown), or a saved configuration. The module in the example is for the currently running kernel, so *static* means that the module is statically bound into the kernel and changing this state would require relinking the kernel executable and rebooting. The module could also be in the *unused* state, which means it is installed but not used, the *loaded* state, which means it has been dynamically loaded into the kernel, or the *auto* state, which means it will be dynamically loaded when it is first needed but hasn't been loaded yet.

The following list shows commonly used options to **kcmodule**:

kcmodule command-line flags:

-a	Includes all modules in the output.
-B	Backs up the currently running configuration prior to changing it.
-c config	Specifies the saved configuration to manage. If none is specified, manage the currently running configuration.
-C comment	Includes a comment pertaining to this invocation of the command in the kernel configuration log file.
-d	Adds the description for each item.
-D	Displays elements for which there is a pending change at the next boot.
-h	Holds the specified change(s) for the next boot.
-K	Keeps the currently running configuration, but does not back it up. Keep the existing backup unmodified.
-P	Parses using the specified output format.
-S	Displays the elements that have been set to something other than the default.
-v	Displays items using verbose output.

Some of these options will be used in the upcoming example of updating the kernel.

kctune

kctune queries and changes the value of kernel tunable parameters in the currently running kernel configuration or a saved kernel configuration that you are staging for future use.

kctune with no options provides the parameters in your system, as shown in the following abbreviated example:

```
# kctune
Tunable                   Value  Expression   Changes
NSTREVENT                    50  Default
NSTRPUSH                     16  Default
NSTRSCHED                     0  Default
STRCTLSZ                   1024  Default
STRMSGSZ                      0  Default
acctresume                    4  Default
acctsuspend                   2  Default
aio_listio_max              256  Default      Immed
aio_max_ops                2048  Default      Immed
aio_monitor_run_sec          30  Default      Immed
aio_physmem_pct              10  Default
aio_prio_delta_max           20  Default      Immed
aio_proc_thread_pct          70  Default      Immed
aio_proc_threads           1024  Default      Immed
aio_req_per_thread            1  Default      Immed
allocate_fs_swapmap           0  Default
alwaysdump                    0  Default      Immed
bufcache_hash_locks         128  Default
chanq_hash_locks            256  Default
core_addshmem_read            1  1            Immed
core_addshmem_write           1  1            Immed
create_fastlinks              0  Default
dbc_max_pct                  50  Default      Immed
dbc_min_pct                   5  Default      Immed
default_disk_ir               0  Default
disksort_seconds              0  Default
dma32_pool_size       268435456  Default
dmp_rootdev_is_vol            0  Default
dmp_swapdev_is_vol            0  Default
dnlc_hash_locks             512  Default
dontdump                      0  Default      Immed
dst                           1  Default
dump_compress_on              1  Default      Immed
enable_idds                   0  Default      Immed
eqmemsize                    15  Default
executable_stack              0  Default      Immed
fs_async                      0  Default
fs_symlinks                  20  Default      Immed
ftable_hash_locks            64  Default
hp_hfs_mtra_enabled           1  Default
io_ports_hash_locks          64  Default
ksi_alloc_max             33600  Default      Immed
ksi_send_max                 32  Default
max_acct_file_size       2560000  Default      Immed
max_async_ports              50  Default
max_mem_window                0  Default
max_thread_proc             256  Default      Immed
```

```
maxdsiz                1073741824  Default      Immed
maxdsiz_64bit          4294967296  Default      Immed
maxfiles                     8192  8192
maxfiles_lim                 8192  8192         Immed
maxrsessiz                8388608  Default
maxrsessiz_64bit          8388608  Default
maxssiz                   8388608  Default      Immed
maxssiz_64bit           268435456  Default      Immed
maxtsiz                 100663296  Default      Immed
maxtsiz_64bit          1073741824  Default      Immed
maxuprc                      256  Default      Immed
maxvgs                        10  Default
msgmap                      1026  Default
msgmax                      8192  Default      Immed
msgmnb                     16384  Default      Immed
msgmni                       512  Default
msgseg                      8192  Default
msgssz                        96  Default
msgtql                      1024  Default
ncdnode                      150  Default      Immed
nclist                      8292  Default
ncsize                      8976  Default
nfile                      65536  Default      Auto
nflocks                     4096  Default      Auto
ninode                      4880  Default

                .
                .
                .
```

This output shows the tunable parameter, its current value, the expressions used to compute the value (which is the default in all cases in the example except for maxfiles), and changes to the value if any are pending. In an upcoming example, I modify the nproc tunable.

Using the -d option, which also works with **kcmodule**, adds a description for each parameter, as shown in the following truncated example:

```
# kctune -d
Tunable                    Value  Expression   Changes
    Description
NSTREVENT                     50  Default
    Maximum number of concurrent Streams bufcalls
NSTRPUSH                      16  Default
    Maximum number of Streams modules in a stream
NSTRSCHED                      0  Default
    Number of Streams scheduler daemons to run (0 = automatic)
STRCTLSZ                    1024  Default
    Maximum size of the control portion of a Streams message (bytes)
STRMSGSZ                       0  Default
    Maximum size of the data portion of a Streams message (bytes; 0 = unlimited)
acctresume                     4  Default
    Relative percentage of free disk space required to resume accounting
acctsuspend                    2  Default
    Relative percentage of free disk space below which accounting is suspended
aio_listio_max               256  Default      Immed
    Maximum number of async IO operations that can be specified in lio_list call
aio_max_ops                 2048  Default      Immed
    Maximum number of async IO operations that can be queued at any time
aio_monitor_run_sec           30  Default      Immed
    Frequency of AIO Thread Pool Monitor Execution (in seconds)
aio_physmem_pct               10  Default

                .
                .
                .
```

Each module now has a more detailed description associated with it as a result of using the -d option.

To group parameters based on the kernel module that defines the tunable, use the -g option, as shown in the following abbreviated example:

```
# kctune -g
Module     Tunable                   Value  Expression   Changes
cdfs       ncdnode                     150  Default      Immed
dump       alwaysdump                    0  Default      Immed
dump       dontdump                      0  Default      Immed
dump       dump_compress_on              1  Default      Immed
fs         bufcache_hash_locks         128  Default
fs         dbc_max_pct                  50  Default      Immed
fs         dbc_min_pct                   5  Default      Immed
fs         disksort_seconds              0  Default
fs         dnlc_hash_locks             512  Default
fs         fs_async                      0  Default
fs         fs_symlinks                  20  Default      Immed
fs         ftable_hash_locks            64  Default
fs         maxfiles                   8192  8192
fs         maxfiles_lim               8192  8192         Immed
fs         ncsize                     8976  Default
fs         nfile                     65536  Default      Auto
fs         nflocks                    4096  Default      Auto
fs         o_sync_is_o_dsync             0  Default
fs         sendfile_max                  0  Default
fs         vnode_cd_hash_locks         128  Default
fs         vnode_hash_locks            128  Default
hpstreams  NSTREVENT                    50  Default
hpstreams  NSTRPUSH                     16  Default
hpstreams  NSTRSCHED                     0  Default
hpstreams  STRCTLSZ                   1024  Default
hpstreams  STRMSGSZ                      0  Default
hpstreams  streampipes                   0  Default
idds       enable_idds                   0  Default      Immed
inet       tcphashsz                  2048  Default
io         aio_listio_max              256  Default      Immed
io         aio_max_ops                2048  Default      Immed
io         aio_monitor_run_sec          30  Default      Immed
io         aio_physmem_pct              10  Default
io         aio_prio_delta_max           20  Default      Immed
io         aio_proc_thread_pct          70  Default      Immed
io         aio_proc_threads           1024  Default      Immed
io         aio_req_per_thread            1  Default      Immed
io         io_ports_hash_locks          64  Default
io         max_async_ports              50  Default
ite        scroll_lines                100  Default      Immed
lvm        maxvgs                       10  Default
pm         acctresume                    4  Default
pm         acctsuspend                   2  Default
pm         chanq_hash_locks            256  Default
pm         dst                           1  Default
                    .
                    .
                    .
```

This output shows all the tunables grouped with their kernel modules. You can see that, in the case of the *fs* module, for example, many tunables are associated with some modules.

The -v output, as shown in the following abbreviated example, provides a lot of tunable-related information:

```
# kctune -v
Tunable             NSTREVENT
Description         Maximum number of concurrent Streams bufcalls
Module              hpstreams
Current Value       50 [Default]
Value at Next Boot  50 [Default]
Value at Last Boot  50
Default Value       50
Can Change          At Next Boot Only

Tunable             NSTRPUSH
Description         Maximum number of Streams modules in a stream
Module              hpstreams
Current Value       16 [Default]
Value at Next Boot  16 [Default]
Value at Last Boot  16
Default Value       16
Can Change          At Next Boot Only

Tunable             NSTRSCHED
Description         Number of Streams scheduler daemons to run (0 = automatic)
Module              hpstreams
Current Value       0 [Default]
Value at Next Boot  0 [Default]
Value at Last Boot  0
Default Value       0
Can Change          At Next Boot Only

Tunable             STRCTLSZ
Description         Maximum size of the control portion of a Streams message (by
tes)
Module              hpstreams
Current Value       1024 [Default]
Value at Next Boot  1024 [Default]
Value at Last Boot  1024
Default Value       1024
Can Change          At Next Boot Only

Tunable             STRMSGSZ
Description         Maximum size of the data portion of a Streams message (bytes
; 0 = unlimited)
Module              hpstreams
Current Value       0 [Default]
Value at Next Boot  0 [Default]
Value at Last Boot  0
Default Value       0
Can Change          At Next Boot Only
Standard input

                    .
                    .
                    .
```

This output shows additional information, including the value the parameter will have upon the next boot.

All tunables have manual pages. If, for example, you want to know more about a tunable, just issue the **man** for the tunable, as shown in the following example for *nproc*:

```
# man nproc

 nproc(5)                                                          nproc(5)
                          Tunable Kernel Parameters

 NAME
      nproc - limits the number of processes allowed to exist simultaneously

 VALUES
    Failsafe
      4200

    Default
      4200

    Allowed values
      100 - 30000

      This may be set higher, but more will not be used.  Setting nproc
      below 110 will interfere with the systems ability to execute in
      multi-user mode.  Some configurations may have a higher minimum.

      nproc must be greater than nkthread + 100.

Standard input
```

Only the beginning of the *nproc* output is shown, but you can see there is a lot of useful information in the man page, such as the default and allowed values. The following list shows commonly used options to **kctune**.

kctune command-line flags:

-a	Includes all information in the output that is normally suppressed.
-B	Backs up the currently running configuration prior to changing it.
-c config	Specifies the saved configuration to manage. If none is specified, manages the currently running configuration.
-C comment	Includes a comment pertaining to this invocation of the command in the kernel configuration log file.
-d	Displays the descriptions of each item.
-D	Displays elements for which there is a pending change at the next boot.
-g	Groups related tunables in the output.
-h	Holds the specified change(s) for the next boot.

-K	Keeps the currently running configuration, but does not back it up. Keeps the existing backup unmodified.
-P	Parses using the specified output format.
-S	Displays the elements that have been set to something other than the default.
-u	Allows the creation of user-defined tunables.
-v	Displays items using verbose output.

Some of these options are used in the upcoming kernel configuration example.

kconfig

kconfig manages kernel configurations.

Running **kconfig** with no options shows you all the saved kernel configurations on your system. You can view the output using -a for all, -v for verbose, and -P for parse.

Changes to the kernel can be delayed until the next boot using the -n option to **kconfig**. (Next boot options are also available with the -h option for **kcmodule** and **kctune**.)

To obtain a list of changes being held for the next boot, you would use the -D option to **kconfig** to show differences between the currently running kernel and what is planned for the next boot. You could also run **kcmodule** and **kctune** with the -D option to get this same list. If you don't want these changes to be applied at the next boot, use the -H option to **kconfig**, which "unholds" the settings.

To obtain a list of non-default kernel values, use the -S option to **kconfig**. This too is a shortcut for running both **kcmodule** and **kctune** with the -S option.

You can specify a saved configuration with no option to **kconfig** by just naming it on the command line or by using the -c option for **kcmodule** and **kctune**.

If you have made changes to the currently running kernel and want to save, use the -s option to **kconfig**.

You can load a saved configuration using the -l option to **kconfig**. If the configuration can be loaded without a reboot, the change takes effect imme-

diately. If not, the change is held for the next reboot. You can specify that the change be applied at the next reboot with the -n option for **kconfig**. You can identify the configuration to be loaded a the next boot using the -w option to **kconfig**.

Several of these options are used in the upcoming kernel reconfiguration example.

The following is a list of commonly used **kconfig** options.

kconfig command-line flags:

-a	Includes all information in the output that is normally suppressed. This provides a lot of information so you may want to pipe this to **more**.
-B	Backs up the currently running configuration prior to changing it.
-C comment	Includes a comment pertaining to this invocation of the command in the kernel configuration log file.
-d config	Deletes the specified kernel configuration.
-D	Displays elements for which there is a pending change at the next boot.
-e	Exports the saved configuration.
-h	Holds the specified change(s) for the next boot.
-H	Discards all changes being held for the next boot.
-i config	Imports the specified configuration.
-K	Keeps the currently running configuration, but does not back it up. Keeps the existing backup unmodified.
-l config	Loads the specified configuration.
-P	Parses using the specified output format.
-S	Displays the elements that have been set to something other than the default.
-v	Displays items using verbose output.

kclog

kclog manages the kernel configuration log file.

All the commands previously covered (**kcmodule**, **kctune**, and **kconfig**) update and maintain the kernel configuration plain text log file called **/var/adm/kc.log**. You can view this file directly to see the kernel-related commands that have been issued, which is the way that I view kernel-related changes, or issue the **kclog** command to view **/var/adm/kc.log**.

kclog has the following commonly used command-line flags:

kclog commonly used command line options:

-a	Prints all entries matching
-c config	Prints log file entries from the specified configuration.
-C comment	Includes the specified comment.

kcusage

kcusage shows the usage level of kernel resources. If you issue the **kcusage** command with no options, you get output for the currently running system, as shown in the following output:

```
# kcusage
Tunable                 Usage / Setting
============================================
dbc_max_pct                 5 / 50
maxdsiz              37666816 / 1073741824
maxdsiz_64bit         7258112 / 4294967296
maxfiles_lim               56 / 8192
maxssiz              1179648 / 8388608
maxssiz_64bit          20480 / 268435456
maxtsiz               421888 / 100663296
maxtsiz_64bit         237568 / 1073741824
maxuprc                     0 / 256
max_thread_proc            57 / 256
msgmni                      2 / 512
msgseg                      0 / 8192
msgtql                      0 / 1024
nfile                     586 / 65536
```

```
nflocks                    20 / 4096
ninode                    653 / 4880
nkthread                  484 / 8416
nproc                     150 / 4200
npty                        0 / 60
nstrpty                     0 / 60
nstrtel                     0 / 60
semmni                     23 / 2048
semmns                     25 / 8192
shmmax                17906400 / 68719476736
shmmni                      7 / 400
[rx8620b{root}:/roothome]>
```

This is an idle system so the resources are used minimally.

You can specify the time period over which data should be printed, including 24 hours, 31 days, and 52 weeks. Two interesting options to **kcusage** are -l, which prints a long format, and -t, which prints the top 5 users or processes that have consumed each resource. The following is an obviated listing of these two outputs:

```
# kcusage -l
Parameter:      dbc_max_pct
Usage:          5
Setting:        50
Percentage:     10.0

Parameter:      maxdsiz
Usage:          37666816
Setting:        1073741824
Percentage:     3.5

Parameter:      maxdsiz_64bit
Usage:          7258112
Setting:        4294967296
Percentage:     0.2

Parameter:      maxfiles_lim
Usage:          56
Setting:        8192
Percentage:     0.7

Parameter:      maxssiz
Usage:          1179648
Setting:        8388608
Standard input
                  .
                  .
                  .
```

```
# kcusage -t
Tunable                  Usage / Setting           Usage      Id Name
================================================================================
dbc_max_pct                  5 / 50
maxdsiz               37666816 / 1073741824
                                                37666816    3009 java
                                                28766208    1278 prm3d
                                                26869760    1254 midaemon
                                                 4468736    1289 scopeux
                                                 2424832    2438 rep_server
maxdsiz_64bit          7258112 / 4294967296
                                                 7258112    1125 cimserver
                                                 2019328     383 utmpd
                                                 1814528    1506 icodd
                                                 1134592     501 ipmon
                                                   69632    1505 cimserverd
maxfiles_lim                56 / 8192
```

```
                                                    56    3009 java
                                                    30     757 inetd
                                                    22    1156 pwgrd
                                                    20    1289 scopeux
                                                    20    2439 agdbserver
maxssiz              1179648 / 8388608
                                                1179648    3009 java
                    .
                    .
                    .
```

kcusage has the following commonly used command-line flags:

-h	Prints kernel usage data over the past hour in 5-minute intervals
-d	Prints kernel usage data over the past 24 hours in hourly intervals
-m	Prints kernel usage data over the past 31 days in daily intervals
-y	Prints kernel usage data for the past 52 weeks in weekly intervals
-l	Prints the listing in long format
-t	Prints a listing that includes the top 5 processes or users of each resource

kcalarm

kcalarm manages alarms of kernel tunable parameters. By using **kcalarm**, you can perform a variety of alarm-related tasks.

kcalarm has the following command-line flags:

-a	Adds a tunable alarm.
-d	Deletes a tunable alarm.
-F	Forces a change in the status of an alarm.

-t threshold	Sets the threshold, which is based on a percentage of the current tunable value, such as 80%, which is the default
-e	Specifies the type of event that will trigger an alarm, such as *initial*, *repeat*, or *return*
-i interval	Specifies the sampling interval of the tunable in minutes. The default is 5 minutes
-c comment	Identifies the alarm request with your comment
-k key	Specifies a key that makes clear the alarm
-n notification	Target to be notified if the alarm is triggered such as the email address to which an email will be sent or a **syslog** entry to be written
-l	Produces a long listing
-s(on|off)	Sets the alarm on or off
-m(on|off|status)	Sets the monitoring of the kernel tunable as on, off, or view its present status

kcmond

The **kcmond** daemon monitors the amount of kernel resources consumed as part of Event Monitoring Service (EMS.) The data that **kcmond** captures can be displayed with **kcusage**. **kcmond** is an important part of managing alarms with **kcalarm** as described earlier. **kcmond** is started as part of EMS and is not designed to be run from the command line.

Building a Kernel

This section shows an example of taking an existing kernel running on an Integrity (Itanium) server and making some changes to it. Figure 5-1 shows some commonly performed kernel-related steps. I perform many of these steps in the following example.

Goal	Command
Save existing configuration with comment in **/var/adm/kc.log** and **/stand** directory name	**kconfig** -C "comment" -**s** *dir_name* (*dir_name* is in **/stand**)
↓	↓
Check existing tunable values with **kctune** and **kcusage**	**kctune** *tunable_name* **kcusage** (-l for long or –t for top 5)
↓	↓
Set tunable parameters with **kctune**	**kctune** -C "comment in kc.log" *name=value*
↓	↓
Check existing modules with **kcmodule**	**kcmodule** -**d** *module_name*
↓	↓
Modify module values with **kcmodule**	**kcmodule** -**C** "comment in **kc.log**" *name=value*
↓	↓
Display differences being held for next book with **kconfig**	**kconfig** -**D**
↓	↓
Reboot when ready	**shutdown** -**r now**
↓	↓
Save new configuration with comment in **kc.log** and **/stand** directory name	**kconfig** – **C** "comment in kc.log" -**s** *dir_name*
Additional Considerations	
Give a title to saved configuration	**kconfig** -**t** *configuration "comment"*
To view **/var/adm/kc.log** either view file or use **kclog** command	**kc.log** -*number_to_view*
To revert to saved configuration	HPUX> **boot** *config_name* (Integrity) ISL> **hpux** *config_name* **vmunix** (HP 9000)

Figure 5-1 Commonly Performed Kernel Configuration Steps

Normally, you should change the kernel configuration before making any changes to it. To save the configuration, use -s and place a comment with -C as shown in the following command:

```
# kconfig -C "Initial configuration of vPars/Oracle/Peoplesoft system" -s
              Initial_vPars_Oracle_Peoplesoft
       * The current configuration has been saved to
         'Initial_vPars_Oracle_Peoplesoft'.
#
```

This command results in the following entry being made in **/var/adm/kc.log**:

```
=============================================================================

Change to configuration 'Initial_vPars_Oracle_Peoplesoft'
at 09:38:45 EDT on 21 September 2004 by root:
Configuration saved from currently running configuration.

Initial configuration of vPars/Oracle/Peoplesoft system
```

This is the last entry in **/var/adm/kc.log** immediately after the **kconfig** command was issued. The -s option results in this kernel having been saved in the directory **/stand/Initial_vPars_Oracle_Peoplesoft**, as shown in the following long listing of **/stand**:

```
Initial configuration of vPars/Oracle/Peoplesoft system

# ll /stand
total 96976
drwxr-xr-x   5 root       sys   8192 Sep 21 09:38 Initial_vPars_Oracle_Peoplesoft
drwxr-xr-x   5 root       sys   8192 Sep 21 09:11 backup
dr-xr-xr-x   3 bin        bin     96 May  5 15:46 boot.sys
-rw-r--r--   1 root       sys     21 May  5 15:44 bootconf
lrwxr-xr-x   1 root       root    14 Sep 21 09:17 bootfs -> current/bootfs
drwxr-xr-x   5 root       root  8192 May 25 12:03 crashconfig
drwxr-xr-x   5 root       sys   8192 Sep 21 09:11 current
drwxr-xr-x   5 root       sys   8192 Sep 19 15:38 initial_11iv2
drwxr-xr-x   5 root       sys   8192 Sep 12 20:28 installed
-rw-r--r--   1 root       sys  16024 Sep 21 09:17 ioconfig
-r--r--r--   1 root       sys     82 May  5 16:11 kernrel
drwxr-xr-x   2 root       sys     96 Sep 21 09:21 krs
drwxr-xr-x   5 root       sys   8192 May  5 16:12 last_install
drwxr-xr-x   2 root       root    96 May  5 15:43 lost+found
lrwxr-xr-x   1 root       root     7 Sep 21 09:17 nextboot -> current
-rw-------   1 root       root    12 Sep 21 09:17 rootconf
lrwxr-xr-x   1 root       root    15 Sep 21 09:17 system -> nextboot/system
-r--r--r--   1 root       sys   1996 May  5 16:01 system.import
-rw-r--r--   1 root       sys   2537 May  5 16:51 system.prev
-rwxr-xr-x   5 root       other 4656 Jun 15 15:49 vmunix
#
```

This directory contains all files associated with the currently running kernel, as shown in the following long listing which are in the directory **Initial_vPars_Oracle_Peoplesoft**:

```
# ll
total 96784
-rw-r--r--   1 root       sys              0 Sep 21 09:38 .config
-rw-r--r--   1 root       sys            147 Sep 21 09:38 README
drwxr-xr-x   3 root       sys             96 Sep 21 09:38 bootfs
drwxr-xr-x   2 root       sys             96 Sep 21 09:38 krs
drwxr-xr-x   2 root       sys             96 Sep 21 09:38 mod
-rw-r--r--   1 root       sys           2877 Sep 21 09:38 system
-rwxr-xr-x   5 root       other     49534656 Jun 15 15:49 vmunix
#
```

This directory contains the currently running kernel **vmunix**, the current **system** file, and other files associated with the currently running kernel. You may also want to give each configuration a title using the -t option to **kconfig**.

At this point I am protected in that if I had to revert to the saved kernel configuration after making changes I can do so. Now that the current configuration has been saved I will view an existing tunable parameter and module that I plan to change with the following commands:

```
# kctune -d nproc
Tunable  Value  Expression  Changes
    Description
nproc     4200  Default     Immed
    Maximum number of processes on the system
```

```
# kcmodule -d idds
Module  State   Cause
    Description
idds    unused
    Intrusion Detection Data Source
#
```

I also issue **kcusage** on a system before making a change to see the level of usage. If the system is not in use, however, and you're preparing it for use, the usage will be low so the output is not meaningful.

In this example, I change *nproc* from 4200 to 8020 and change *idds* to best. Immediately before making these changes I add a comment to **/var/adm/kc.log** so that a record exists of why these changes were made:

```
# kctune -C "first of many tunable changes for vPars/Oracle/Peoplesoft config"
  nproc=8020

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? y
       * The automatic 'backup' configuration has been updated.
       * The requested changes have been applied to the currently
         running system.
Tunable            Value  Expression  Changes
nproc    (before)   4200  Default     Immed
         (now)      8020  8020
#



# kcmodule -C "first of many module changes for vPars/Oracle/Peoplesoft
               config" idds=best

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 configuration is supposed to include a module 'idds' which
         is not available without a kernel rebuild.
       * The automatic 'backup' configuration has been updated.
       * Building a new kernel for configuration 'nextboot'...
       * Adding version information to new kernel...
       * The requested changes have been saved, and will take effect at
         next boot.
Module               State   Cause
idds    (now)        unused
        (next boot)  static  explicit
#
```

This tunable is dynamic, so the change was made immediately as you can see at the bottom of the listing with *now*. The **kcmodule** command changed *idds*; however, this will be applied at *next boot*, as shown in the following **kconfig** command:

```
# kconfig -D
Module               State   Cause
idds    (now)        unused
        (next boot)  static  explicit
NOTE:    There are no tunable changes being held until next boot.
#
```

In both the **kctune** and **kcmodule** commands, I preceded the change with a comment (-C) that appears on the **/var/adm/kc.log** file.

For the module change to take place, a reboot is required. After the reboot takes place, I again issue the **kcmodule** command to show that the *idds* change has been incorporated into the new current kernel:

```
# kcmodule -d idds
Module  State   Cause
    Description
idds    static  explicit
    Intrusion Detection Data Source
#
```

You would probably make more changes to the kernel than the two used in this example. After the changes are made, you would save the update configuration with the **kconfig** command:

```
# kconfig -C "Rev1 of vPars/Oracle/Peoplesoft kernel configuration"
 -s rev1_vPars_oracle_peoplesoft

       * The current configuration has been saved to
         'rev1_vPars_oracle_peoplesoft'.

# ll /stand
total 96992
drwxr-xr-x   5 root   sys     8192 Sep 21 09:38 Initial_vPars_Oracle_Peoplesoft
drwxr-xr-x   5 root   sys     8192 Sep 21 10:34 backup
dr-xr-xr-x   3 bin    bin       96 May  5 15:46 boot.sys
-rw-r--r--   1 root   sys       21 May  5 15:44 bootconf
lrwxr-xr-x   1 root   root      14 Sep 21 10:51 bootfs -> current/bootfs
drwxr-xr-x   5 root   root    8192 May 25 12:03 crashconfig
drwxr-xr-x   5 root   sys     8192 Sep 21 10:34 current
drwxr-xr-x   5 root   sys     8192 Sep 19 15:38 initial_11iv2
drwxr-xr-x   5 root   sys     8192 Sep 12 20:28 installed
-rw-r--r--   1 root   sys    16024 Sep 21 10:52 ioconfig
-r--r--r--   1 root   sys       82 May  5 16:11 kernrel
drwxr-xr-x   2 root   sys       96 Sep 21 10:56 krs
drwxr-xr-x   5 root   sys     8192 May  5 16:12 last_install
drwxr-xr-x   2 root   root      96 May  5 15:43 lost+found
lrwxr-xr-x   1 root   root       7 Sep 21 10:51 nextboot -> current
drwxr-xr-x   5 root   sys     8192 Sep 21 11:06 rev1_vPars_oracle_peoplesoft
-rw-------   1 root   root      12 Sep 21 10:51 rootconf
lrwxr-xr-x   1 root   root      15 Sep 21 10:51 system -> nextboot/system
-r--r--r--   1 root   sys     1996 May  5 16:01 system.import
-rw-r--r--   1 root   sys     2537 May  5 16:51 system.prev
-rwxr-xr-x   3 root   sys 49536344 Sep 21 10:34 vmunix
#
```

The long listing of **/stand** shows the **kconfig** command did indeed save the current kernel configuration under the name specified, beginning with *rev1*. In addition, a new comment in **/var/adm/kc.log** identifies that this step was taken. Now both the original configuration that was saved earlier and the current configuration exist in **/stand**.

I don't think you can save your kernel configuration too often. I sometimes make hundreds of kernel configuration parameter changes when I prepare a new system to run a specific application, so I'm careful to save my kernel before making these many changes. The only problem with saving many kernel configurations is the space consumed in **/stand**. If you run out of space in **/stand** it can be difficult to increase it, so you must keep an eye on the available space in **/stand**.

You may find yourself in a situation where you have to boot a saved kernel configuration, which is covered in the next section.

Reverting to a Saved Kernel Configuration

Although the two configuration changes made in the previous section were trivial if, for any reason, you need to revert to a saved kernel configuration you can do so. The easiest way to do this is to run **kconfig -n** *oldconfig*. An alternative is to load the saved configuration at the time of boot. In the following example, I could revert back to *Initial_vPars_Oracle_Peoplesoft* or *initial_11iv2* which were saved in **/stand**. To revert to this kernel configuration issue the following command at the HP-UX prompt on an Itanium system running EFI:

```
Please select a boot option

    HP-UX Primary Boot: 0/0/0/2/0.6.0
    Acpi(000222F0,0)/Pci(1|0)/Mac(00306E4B9AD9)
    Acpi(000222F0,100)/Pci(1|0)/Mac(00306E4BAA28)
    EFI Shell [Built-in]
    Boot option maintenance menu


    Use ^ and v to change option(s). Use Enter to select an option
Loading.: HP-UX Primary Boot: 0/0/0/2/0.6.0
Starting: HP-UX Primary Boot: 0/0/0/2/0.6.0

(c) Copyright 1990-2003, Hewlett Packard Company.
All rights reserved

HP-UX Boot Loader for IPF  --  Revision 1.73

Press Any Key to interrupt Autoboot
\EFI\HPUX\AUTO ==> boot vmunix
Seconds left till autoboot -   9
   Type 'help' for help

HPUX> boot initial_11iv2
```

From EFI, I select the *HP-UX Primary Boot* and then interrupt the boot. At this point, I enter *initial_11iv2* as the configuration to load.

On a PA-RISC system, this old configuration would have been entered at the ISL prompt with ISL> **hpux** *config_name* **vmunix**.

System File

Although the kernel commands covered support modifying your configuration and building a new kernel, you can still use **/stand/system** or the **system** file for any kernel configuration. There is a **system** file in the directory

for all the kernel configurations that you've built on your system. These can be exported from one system and imported to another system in order to replicate a kernel configuration that you like onto another system. You can, for example, export a configuration with **kconfig -e** *name* on one system, import a configuration on another system with **kconfig -i** *name,* and thereby replicate a configuration on another system.

Device bindings are also managed in the system files. Device bindings are configuration settings that have to do with specific hardware devices such as primary swap (*swap* lines in the system file), dump devices (*dump* in the system file). and device driver (*device* in the system file) specifications.

Another common use of system files is to make multiple changes in the system file and have all the changes take effect on the next reboot.

The following is an abbreviated **system** file showing some of the lines in the currently running **system** file:

```
# example system file

* Module entries
*
module prm best [3F56E2F0]
module ipf loaded 0.1.0
module mpt best [40075F90]
module vols best [3F41B706]
                    .
                    .
                    .
* Swap entries
*
*
* Dump entries
*
dump lvol
*
* Driver binding entries
*
*
* Tunables entries
*
tunable nproc 4200
tunable shmmax 68719476736
tunable semume 512
tunable semmnu 4092
tunable semmns 8192
tunable maxfiles 8192
tunable maxfiles_lim 8192
                    .
                    .
                    .
```

This listing shows module, swap (none present), dump, and tunable entries.

You could update numerous entries in the **system** file. Issue **kconfig -i** *system_file_name* and reboot this system to incorporate the changes incorporated into the running kernel, as shown in the following listing:

```
# kconfig -i /stand/Initial_vPars_Oracle_Peoplesoft/system
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? y
       * The automatic 'backup' configuration has been updated.
NOTE:    The configuration being loaded contains changes that cannot be
         applied immediately:
NOTE:    The changes will be held for next boot.
       * /stand/Initial_vPars_Oracle_Peoplesoft/system has been
         imported.  The changes will take effect at next boot.
[rx8620b{root}:/]>
```

The **kconfig** command updates the currently running kernel. The changes can not be applied immediately, so a reboot is required. The file **/stand/current/system** reflects the changes that you made to the configuration. You can also give a name to the updated configuration so that a new configuration is produced, as shown in the next example.

You can also make changes to **/stand/current/system** and save the configuration to a specific name, as shown the following example:

```
# kconfig -i test2 /stand/current/system
WARNING: The system file was created from the configuration 'nextboot'.
         The import operation is targeted at 'test2'.
     ==> Do you wish to continue? y
       * /stand/current/system has been imported to 'test2'.
      #
```

Prior to running this **kconfig** command, I update **/stand/current/system** with numerous changes. The changes were made immediately and no reboot was required for these dynamic changes to take effect. This **kconfig** command imports the **system** file and saves it to configuration called *test2*. This results in a **test2** directory in **/stand**, as shown in the following listing:

```
# ll /stand/test2
total 96784
-rw-r--r--   1 root       sys              0 Sep 25 11:24 .config
-rw-r--r--   1 root       sys            147 Sep 25 11:24 README
drwxr-xr-x   3 root       sys             96 Sep 25 11:24 bootfs
drwxr-xr-x   2 root       sys             96 Sep 25 11:24 krs
drwxr-xr-x   2 root       sys             96 Sep 25 11:24 mod
-rw-r--r--   1 root       sys           2902 Sep 25 11:24 system
-rwxr-xr-x   5 root       other     49534656 Jun 15 15:49 vmunix
#
```

This results in a saved configuration reflecting the changes that have been made. This configuration has been saved and can be loaded anytime.

The **mk_kernel -s** *system_file_name* command can also build the new kernel. **mk_kernel** is fully supported and is great for those of us who have used this command in past releases of HP-UX.

kcweb

kcweb is one of many Web-based system-administration tools. It can be invoked from SAM or directly in a browser window. In this section you perform a variety of functions through the Web-based interface. In this section we'll perform the following in **kcweb**:

- View kernel parameters
- Get details on a specific kernel parameter in the bottom of the **kcweb** page and the man page
- Modify a dynamic kernel parameter and apply the new value
- Set an alarm to inform you when a kernel parameter exceeds the specified value

At the time of this writing, **kcweb** is invoked from a browser with the URL https://*ip_address*:1188/casey/login.cgi, where *ip_address* is the IP address of the system on which you want to run **kcweb**. Figure 5-2 shows the main screen you see after invoking **kcweb**.

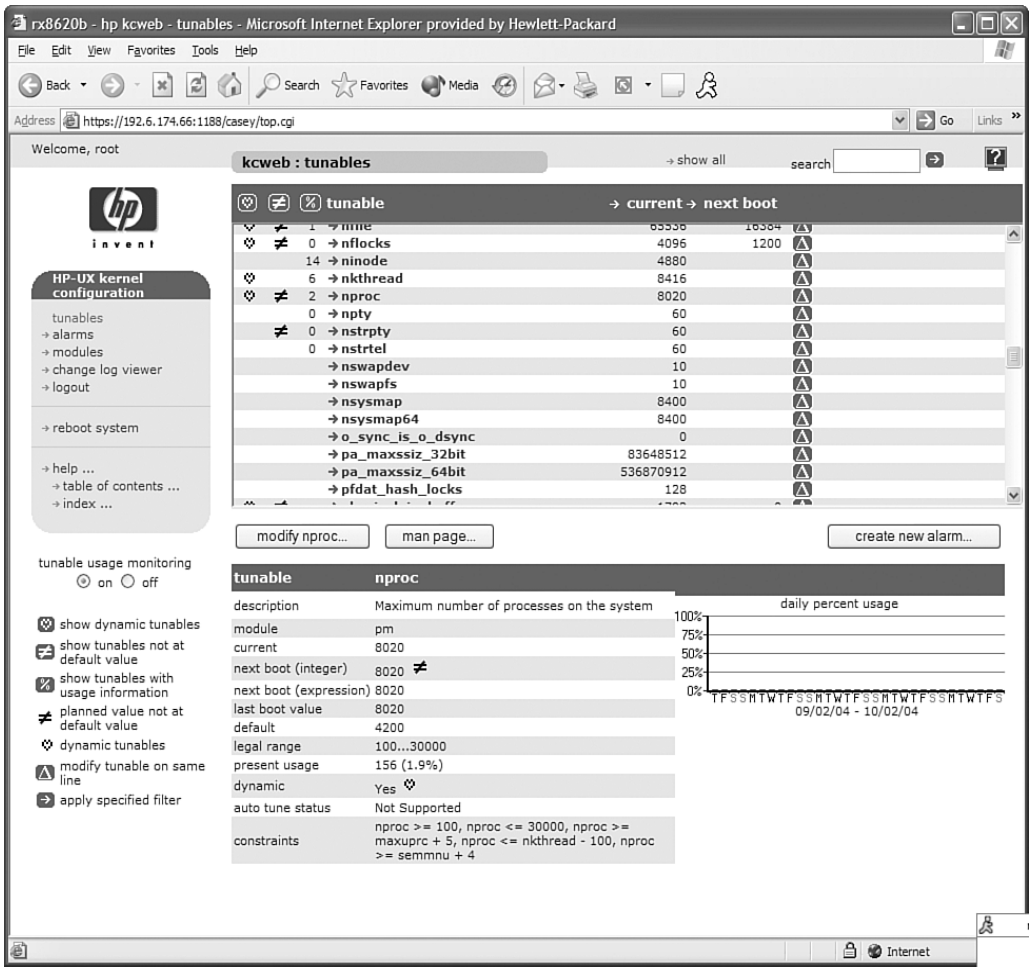

Figure 5-2 **kcweb** Showing *tunables*

The upper left of Figure 5-2 shows that **kcweb** has *tunables, alarms, modules*, and other functions.

Notice that in the bottom left of the figure, there is a legend that includes descriptions of the symbols used in **kcweb**. Those tunables with a heart next to them are dynamically tunable parameters. If you select the heart on the bar across the top of the tunables, only dynamically tunable parameters will be shown. The “not equal to” sign indicates parameters that are not set to their default value. Several other entries exist in the legend as

well. This makes for viewing groups of icons easy and the legend helps identify the status of icons.

The bottom of the screen provides information about the kernel parameter selected: in this case, *nproc*. A graph in the bottom right shows the usage of this parameter over time. For system-wide parameters, the graph shows usage on a system basis. For user-specific or process-specific parameters, the graph includes the top five consumers of the parameter.

The *constraints* at the very bottom of the page give an excellent overview of the tunable. If you need to get detailed information about a kernel parameter, select the *man page...* button, as shown in Figure 5-3.

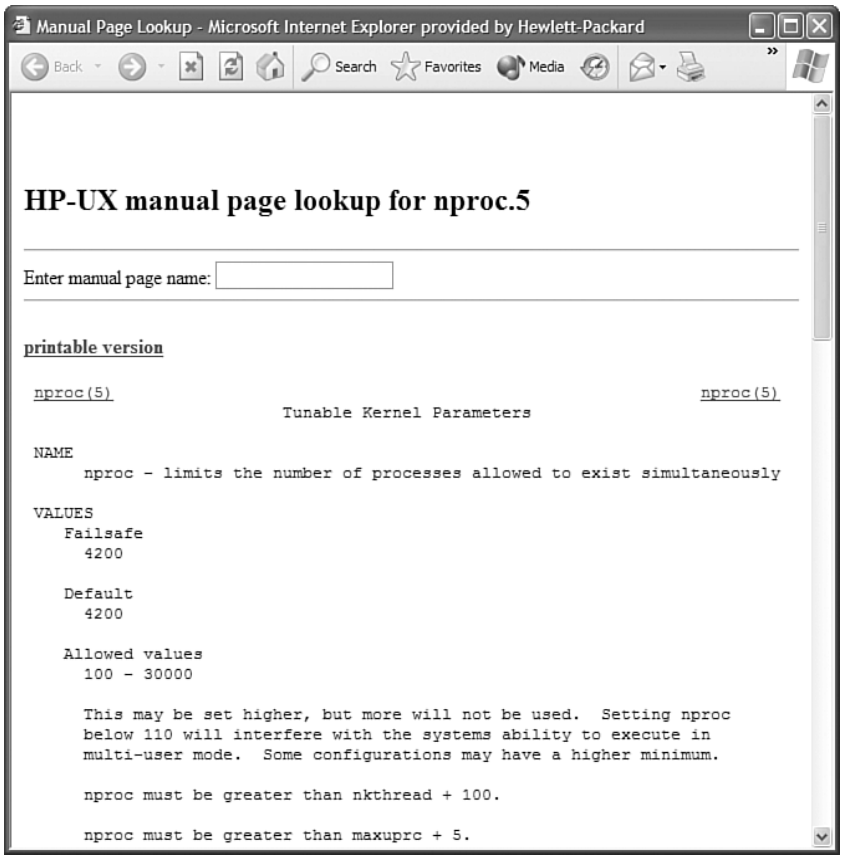

Figure 5-3 **kcweb** Man Page of *nproc*

You can also modify one of the parameters by highlighting the parameter and then selecting *modify <parameter name>* as is done for the *nproc* parameter, as shown in Figure 5-4.

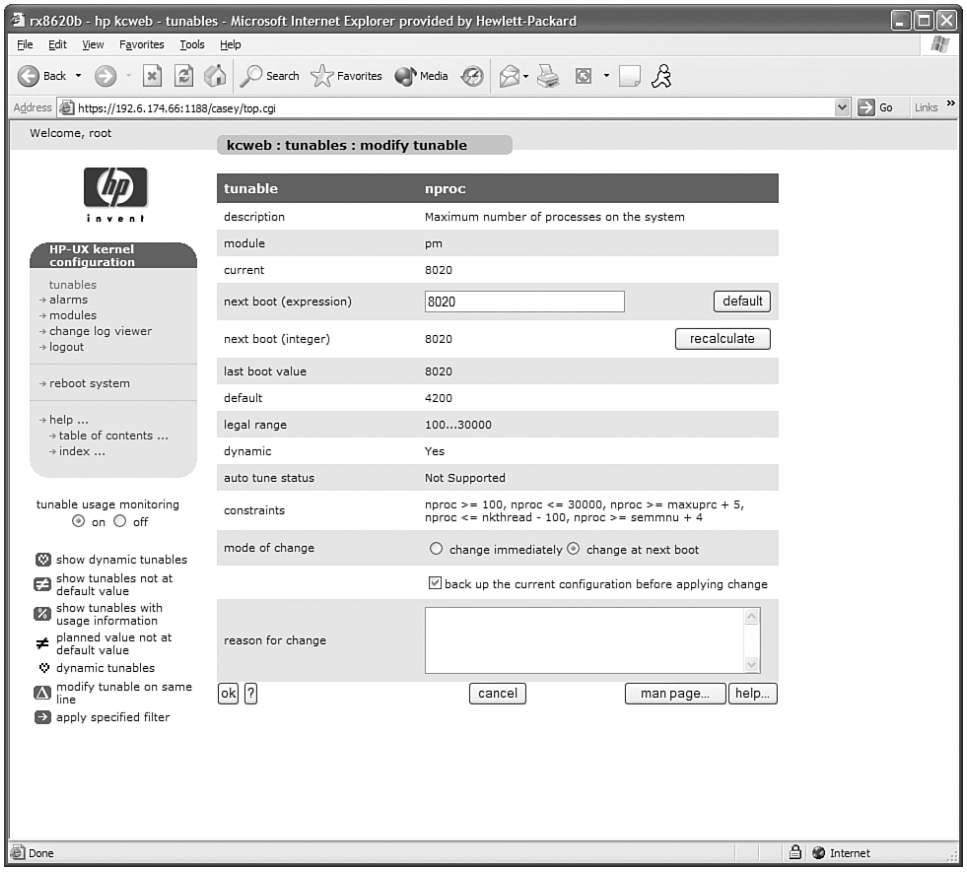

Figure 5-4 **kcweb** Showing *modify nproc*

You can change the value of *nproc* and then select whether you want a change immediately or at the next boot.

You can set an alarm to be informed when the parameter reaches a specified threshold. Figure 5-5 shows the process of setting up this alarm.

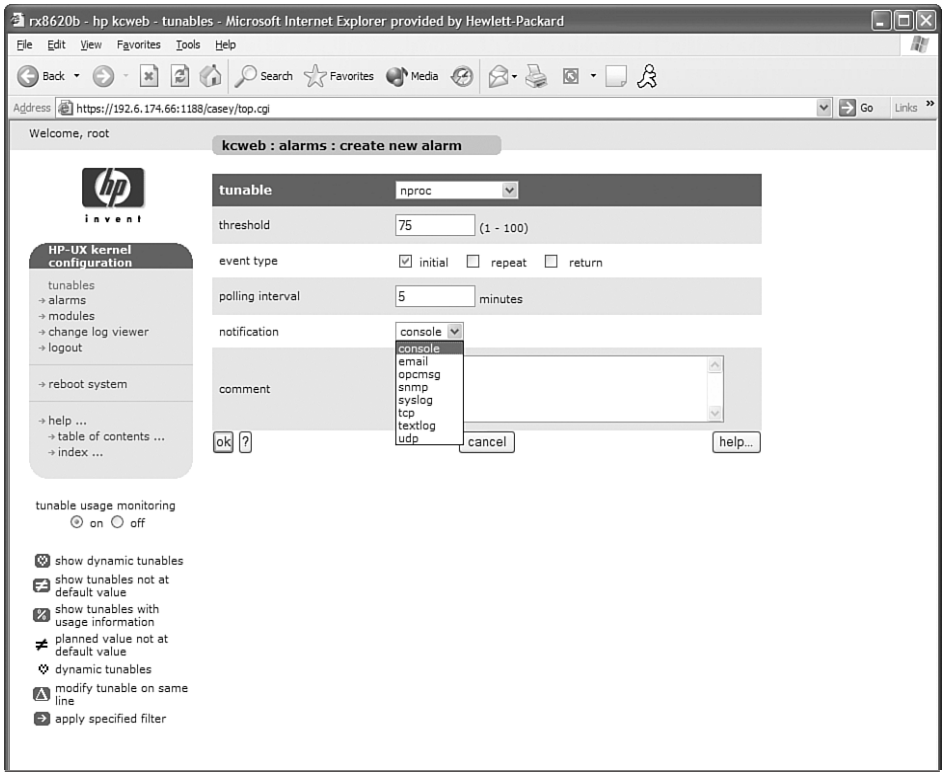

Figure 5-5 **kcweb** Showing *nproc* Alarm

We get to the *kcweb:alarms* page by selecting *create new alarm...* in Figure 5-5. All the parameters related to the alarm are shown in Figure 5-5. The setup of the alarm allows you to specify a *threshold*. All of the options for notification are shown such as sending an email

You can also work with kernel modules in the same way that you work with kernel tunables. Those that are dynamic can be loaded on-the-fly, and those that are not dynamic can be built into the kernel with a rebuild.

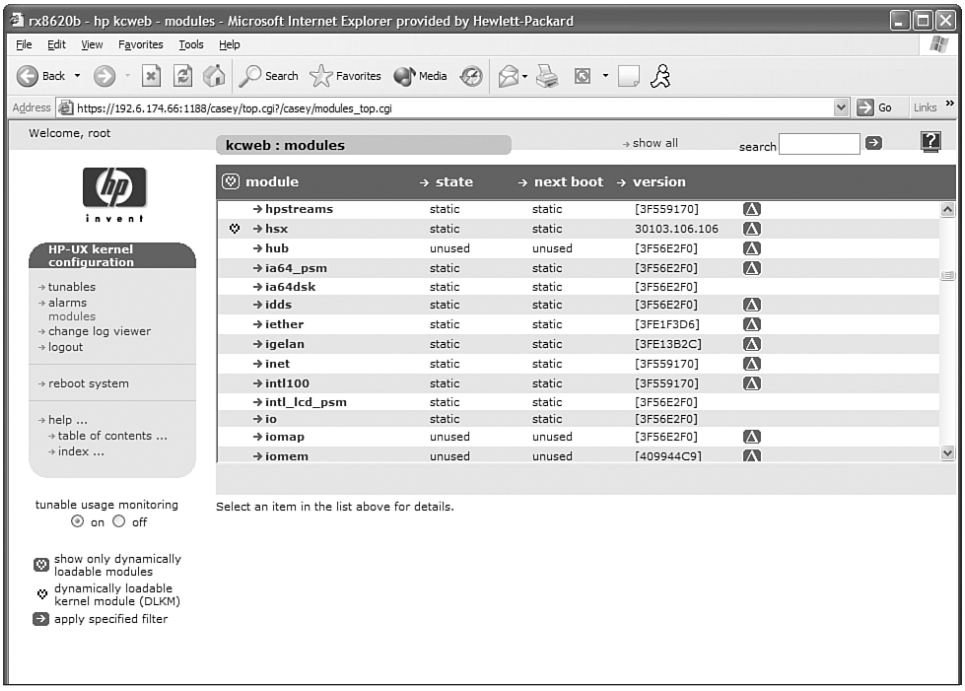

Figure 5-6 **kcweb** Showing *modules*

As with tunables, modules can be dynamic. In Figure 5-6, the idds module that we worked with earlier in the chapter is shown, which is static, as well as hsx, which has a heart next to it to indicate that it is a Dynamically Loadable Kernel Module (DLKM.)

You can also view **/var/adm/kc.log** with *change log viewer,* as shown in Figure 5-7.

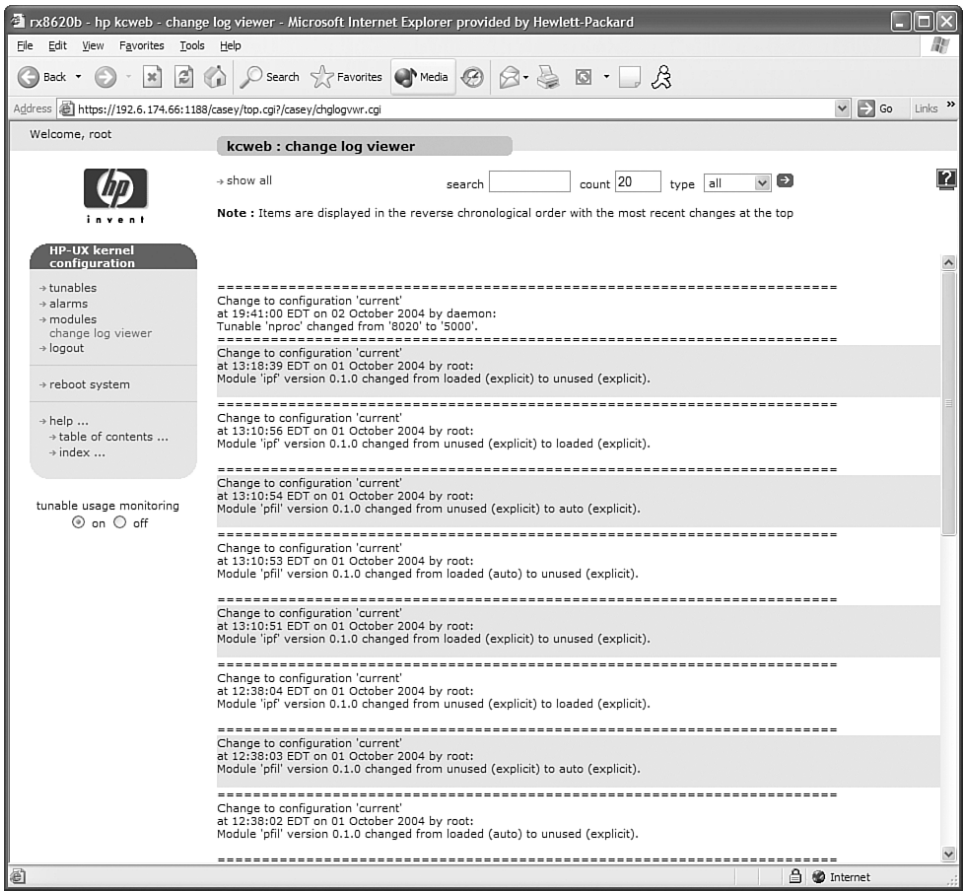

Figure 5-7 **kcweb** Showing Log File

There is also a table of contents and index that you can refer to if you need help with any kernel-related task.

Using **kcweb**, you can accomplish the tasks as you can at the command line. It's a matter of preference whether you take the graphical **kcweb** interface or the command line approach covered earlier in the chapter.

This chapter gave a quick overview of **kcweb**, which included some of the most commonly performed tasks. Because this is a Web-based interface, it is easy to use and most of the screens and information are self-explanatory.

Other Web-based management tools can be invoked through SAM including **pdweb** and **parmgr**; those are covered in other chapters.