Chapter 9
-
Kernel Configuration

INDEX

Introduction

The architecture of the HP-UX kernel has been undergoing changes over serveral releases. The kernel is becoming more modular and more capable of beeing tuned while running.

- At UX 9.X `uxgen` was the tool to compile a fully static kernel file called `hp-ux` based on the system file called `S800`.
- At UX 10.20 `mk_kernel` was the tool to compile the kernel called `/stand/vmunix`, based on the system file call `/stand/system`. The kernel configuration could be performed using SAM.
- At UX 11.00 the KM-commands were introduced that should discourage from editing the system file directly. Additionally loadable module (DLKM) support needed to be integrated into kernel configuration.
- At UX 11.11 dynamical tunables were introduced that could be modified without rebooting the system.
- At UX 11.22 a web gui kcweb was introduced that allows to config the kernel as well as monitor kernel tunable usage.
- At UX 11.23 the concept of saved kernel configurations and new KC-commands were introduced. The kcweb interface was extended.

The next sections describes the kernel configuration features for each HP-UX release and provides a procedure that describes how to tune a kernel.

Kernel Configuration at UX 10.X

Generating a new kernel is done via SAM or with a shell script called `mk_kernel(1M)`. Based on a user-provided **system file** (`/stand/system`) containing the names of the included device drivers and the settings of the kernel tunables, `mk_kernel` builds an executable file which can be used as a bootable kernel.
`mk_kernel(1M)` basically calls a the `config(1M)` command who does the main work.
`config(1M)` reads the system file and the master kernel configuration table information and generates the following output files:

- C program source files (`conf.c` and `space.h`) that define the configuration tables for various parts of the system.
- a makefile (`config.mk`) to compile the C program produced and relink the newly configured system.

`config(1M)` executes the `make(1)` command to compile `conf.c` and link the kernel with the appropriate kernel libraries. Many header files are needed to compile `conf.c`. Also, archive library files containing the kernel objects are needed to link the kernel. These files are supplied with the system and are contained in the directories found under `/usr/conf`. Libraries for the kernel are expected to be found in `/usr/conf/lib`. The master file used is

the composite of files found under `/usr/conf/master.d/`.
If the build succeeds, the newly built binary will be a file named `vmunix_test` located in the build directory `/stand/build`. If another path is used to designate the system file, the build directory will be the present working directory.

NOTE: The bundled C compiler `/usr/ccs/bin/cc` is used for kernel compilation not the ANSI C compiler.

Refer to the Software Development Chapter to learn more about the build process.

The System File /stand/system

The system file is the user-provided description of an HP-UX system. It comprises of three parts:

- Device driver
- Swap & dump devices
- Tunable system parameters

Details regarding the syntax can be found in the `config(1M)` manual page.

Device Driver, Pseudo Driver and Subsystems

Each line has the format

```
drivername
```

Where the driver needs to exist in the kernel configuration table in the directory `/usr/conf/master.d`, i.e a file called *drivername* must exist in that directory.

Example:

```
# grep -i fcms /stand/system
fcms

# ll /usr/conf/master.d/fcms
-r--r--r--   1 bin      bin      5828 Nov 20  1999 /usr/conf/master.d/fcms
```

Refer to the I/O Chapter for common device drivers.

Swap and Dump Devices

`/dev/vg00/lvol2` is the primary swap and dump device by default. If you like to configure a whole disk or a section of a disk as primary swap device, the following line needs to be added to the system file:

```
swap  [hw_path]  [offset]  [blocks]  [options]
```

This is not recommended because you won't get a warning if you mistakenly try to use this swap disk for LVM.

You can also configure dump devices on whole disks or sections. This is also not recommended for the same reasons. The system file should only contain the line (it does by default):

```
dump    lvol
```

This entry is the default and it means that you can configure other dump devices in addition to the primary swap device, /dev/vg00/lvol2, (using `lvlnboot -d`).

I/O Statements for Devices

An I/O statement is used to link a device driver in the kernel with a dedicated HW address. This is necessary for devices that do not use a standard driver (like `spt`)

An I/O statements has the following syntax:

```
driver      hw_path      driver_name
```

Example:

```
driver      10/4/8.1.1      spt
```

Tunable Parameters

This part of the system file holds the settings of the kernel tunables. Tunables that do not appear in the system file keep their default value. Each line contains two fields. The first field represents the name of the tunable and the second represents the corresponding value. You can specify the value either in decimal or hexadecimal. Each line is independent, optional, and written in the following format:

```
parameter_name      number  or  formula
```

Examples:

```
nstrpty         60
msgmax          32768
maxdsiz         0x10000000
nproc           (100*maxusers)
```

Detailed explanations of the kernel tunables including their default, minimum and maximum values can be found on the HP document server:
http://docs.hp.com/hpux/onlinedocs/TKP-90202/TKP-90202.html

Example of a System File

```
* Drivers and Subsystems

CentIf
CharDrv
c700
c720
ccio
cdfs
cio_ca0
 ...
 ...
tpiso
uipc
vxbase
wsio
```

```
* Kernel Device info

dump lvol
driver 10/4/8.1.1 spt

* Tunable parameters

nstrpty         60
msgmax          32768
msgmni          50
msgseg          7168
msgssz          8
msgtql          256
```

Procedure to generate a new kernel at UX 10.X

In order to generate a new kernel follow the steps below:

1) Change to the kernel generation working directory and remove the files that have been left over from the last kernel generation:

   ```
   # cd /stand/build
   # rm –r *
   ```
   (Be careful with this! You need to be in `/stand/build/`)

2) Backup the current system file and kernel:

   ```
   # cp /stand/system /stand/system.prev
   # cp /stand/vmunix /stand/vmunix.prev
   ```

3) Is the system file up to date?

   If you are not sure that the current system file (/stand/system) corresponds to the current kernel (`/stand/vmunix`) then use the `system_prep` script to extract a system file from the current kernel:

   ```
   # /usr/lbin/sysadm/system_prep [-s /stand/system]
   ```

   Compare the new system file to the original one. The `diff` command should not produce any results:

   ```
   # diff /stand/system /stand/system.prev
   ```

4) Modify the system file:

   Modify the system file as needed:

   ```
   # vi /stand/system
   ```

5) Generate the kernel:

   After you modified the system file, you can generate the new kernel using the `mk_kernel` script. The name of the new kernel will be `/stand/build/vmunix_test`, unless explicitly specified through the `-o` option. The `-s` option specifies the system file to use, `/stand/system` is the default:

   ```
   # mk_kernel
   ```

6) Move the new kernel files to `/stand`

   The new kernel will be activated by rebooting the system:

   ```
   # mv /stand/build/vmunix_test /stand/vmunix
   ```

7) Restart the system

   The new kernel will be activated by restarting the system:

```
# cd /
# shutdown –r 0
```

Kernel Configuration changes at UX 11.00

With the introduction of DLKM (Dynamically Loadable Kernel Modules) as of HP-UX 11.00, the kernel now is made of a static part and a number of dynamically loadable modules. The kernel does not only consist of a single file (`/stand/vmunix`), but of multiple files and directories (e.g. `/stand/dlkm/`) that are dependent from each other. The dynamically loadable modules can be activated *online* (i.e without rebooting). They also can be activated by the kernel *on demand* (autoload).

The following files belong to a DLKM:

`/usr/conf/master.d/`*`module_name`*	Master configuration tables for kernel and kernel modules.
`/usr/conf/km.d/`*`module_name/mod.o`*	Source code file (object file).
`/stand/system.d/`*`module_name`*	Kernel module system file.
`/stand/dlkm/system.d/`*`module_name`*	Kernel module system file (copy).
`/stand/dlkm/mod.d/`*`module_name`*	Kernel module loadable image.

1.) Procedure to generate a kernel at UX 11.00

Even though you wish to compile an old fashioned *static* kernel without using DLKMs, the procedure is different from the procedure for UX 10.X. The new command, `kmupdate(1M)`, is used to move the kernel file plus associated files to their appropriate places during system restart.

In order to generate a new kernel follow the steps below:

1) Change to the kernel generation working directory and remove the files that have been left over from the last kernel generation:

```
# cd /stand/build
# rm –r *        (Be careful with this! You need to be in /stand/build/)
```

2) Backup the current system file:

```
# cp /stand/system /stand/system.prev
```

3) Is the system file up to date?

   If you are not sure that the current system file (`/stand/system`) corresponds to the current kernel (`/stand/vmunix`), then use the `system_prep` script to extract a

system file from the current kernel:

```
# /usr/lbin/sysadm/system_prep [-s /stand/system]
```

Compare the new system file to the original one. The diff command should not produce any results:

```
# diff /stand/system /stand/system.prev
```

4) Modify the system file

Use `kmtune(1M)` to modify kernel tunables. Use `kmsystem(1M)` to add/remove driver. These commands do nothing more than modifying `/stand/system`. Due to various dependencies, the system file should not be edited by hand. E.g.:

`# kmtune –s dbc_max_pct=35`	(changes tunable in system files)
`# kmsystem –c y diag2`	(adds driver `diag2` to system file)
`# kmsystem –c n diag2`	(removes driver `diag2` from system file)

The „km“ commands are explained below in greater detail.

5) Generate the kernel

After you modified the system file, you can generate the new kernel using the mk_kernel script. The name of the new kernel will be `/stand/build/vmunix_test`, unless explicitly specified through the `-o` option. The `-s` option specifies the system file to use, `/stand/system` is the default:

```
# mk_kernel
```

6) Schedule kernel update

kmupdate is used to initiate the move of the new kernel (`/stand/build/vmunix_test`) and the corresponding files to their appropriate places at the next shutdown or startup. The current kernel will be copied to `/stand/vmunix.prev`, then.

```
# kmupdate
```

7) Restart the system

The new kernel will be activated by restarting the system:

```
# cd /
# shutdown –r 0
```

NOTE:
Do not use the `reboot` command. Otherwise the movement of the kernel files will be skipped and the system boots from the original kernel again. If you really need to reboot instead of shutdown, then execute the rc script manually before rebooting:

```
# /sbin/init.d/kmbuild stop
```

overview about standard kernel files:

`/stand/vmunix`	kernel executable

/stand/vmunix.prev	kernel executable (backup)
/stand/system	system file
/stand/system.prev	system file (backup)
/stand/dlkm/	kernel function set directory
/stand/dlkm.vmunix.prev/	kernel function set directory (backup)
/stand/build/dlkm.vmunix_test/	kernel function set directory (built by mk_kernel)
/stand/build/vmunix_test	new kernel executable (built by mk_kernel)

2.) Add a DLKM to the kernel

To install and activate a DLKM to the running kernel, you need the commands:

kminstall(1M)	(installation of a module) and
kmadmin(1M)	(activation of a module)

Generally a DLKM consists of three components:

code file	/usr/conf/km.d/*module_name*/mod.o
master file	/usr/conf/master.d/*module_name*
system file	/stand/system.d/*module_name*

Activating a DLKM

This example shows the installation and (de)activation of the EMS monitor “krm“ (kernel resource monitor). The krmond daemon is a hardware monitor that is part of the EMS (Event Monitoring Services) subsystem. It simply monitors the usage of kernel tunables. See http://docs.hp.com/hpux/onlinedocs/diag/ems/emd_kern.htm for details regarding krm.

Check if krm is installed:

```
# swlist -l product EMS-KRMonitor
# Initializing...

  EMS-KRMonitor        A.11.11.04     EMS Kernel Resource Monitor
```

The product is not on the application CD (DART), but on the Support Plus CD together with the standard patch bundles. It is bundled with the Diagnostics bundle (OnlineDiag). See http://software.hp.com/ under *Enhancement Releases*.

It contains the following files:

/dev/krm	driver
/etc/opt/resmon/lib/krm/mod.o	code file
/master	master file
/system	system file
/etc/opt/resmon/lbin/krmond	daemon
/opt/resmon/share/man/man1m/krmond.1m	manual page
/etc/opt/resmon/dictionary/krmond.dict	dictionary file

In order to install the module, you need to copy the files (subsystem code, master and system file) to the correct places by using kminstall:

```
# cd /etc/opt/resmon/lib/krm
# kminstall -a krm
```

subsystem code to	`/usr/conf/km.d/`**`krm/mod.o`**
master file to	`/usr/conf/master.d/`**`krm`**
system file to	`/stand/system.d/`**`krm`**

Use `kmsystem` to check if the installation of the module was successful:

```
# kmsystem -q krm
Module              Configured          Loadable
================================================
krm                 Y                   Y
```

The command `mk_kernel` is used to generate a new kernel module, i.e subsystem code, master and system file are combined into a single usable kernel module `/stand/dlkm/mod.d/`*`krm`*..
A copy of the system file will be placed in `/stand/dlkm/system.d/`*`krm`*.
Internally, the commands `config -M krm` and `kmupdate -M krm` are being executed.

```
# mk_kernel -M krm
Generating module: krm...

  Specified module(s) below is(are) activated successfully.
      krm

WARNING: Kernel update request is scheduled.
         Activated module(s) is(are) only available until system reboot.
```

`kmadmin` activates the module, i.e it loads it into the running kernel.

```
# kmadmin -L krm
kmadmin: Module krm loaded, ID = 1
```

`kmadmin` is also useful to check whether the module has been loaded successfully:

```
# kmadmin -Q krm
Module Name             krm
Module ID               1
Module Path             /stand/dlkm/mod.d/krm
Status                  LOADED
Size                    8192
Base Address            0x429c000
BSS Size                0
BSS Base Address        0x0
Hold Count              1
Dependent Count         0
Unload Delay            0 seconds
Description             krm
Type                    WSIO
Block Major             -1
Character Major         240
Flags                   a5
```

The command

```
kminstall -d krm
```

removes the module again, i.e. the status is like before `kminstall -a` was executed.

Overview of the DLKM commands

The „km“ commands can be found in the directory `/usr/sbin`

kminstall(1M)

Add/remove/update a kernel module. kminstall(1M) adds or removes the module's subsystem code, master and system file.

Removing the krm module:

```
# kminstall -d krm
# kmsystem -q krm
kmsystem: Invalid module, subsystem or driver name : krm
# cat /stand/system.d/krm
cat: Cannot open /stand/system.d/krm: No such file or directory
```

Adding the krm module again:

```
# kminstall -a krm
kminstall: Local directory must contain a mod.o file
# cd /etc/opt/resmon/lib/krm
# kminstall -a krm
# cat /stand/system.d/krm
* @(#)$Revision: 1.1 $ $Date: 2000-04-18 11:18:16-06 $
* Kernel Resource Monitor system file
* (C) Copyright 1999 Hewlett-Packard Company
$VERSION        1
$CONFIGURE      Y
```

The update option (-u) is neccessary whenever the subsystem code (mod.o) has changed.

Refer to kminstall(1M) man page for details

kmsystem(1M)

Displays/modifies the LOADABLE and CONFIGURATION flags for kernel modules. The LOADABLE flag determines whether the module gets loaded dynamically to the kernel or whether it gets statically linked to the kernel. The CONFIGURATION flag determines whether the module should be included during the next kernel compilation or not. Static drivers are added to/removed from the system file with this option.
kmsystem(1M) simply modifies the module's system file `/stand/system.d/<module>`

```
# kmsystem | more
Module              Configured          Loadable
================================================
CentIf              N                   -
CharDrv             N                   -
DlkmDrv             Y                   -
GSCtoPCI            Y                   -
PCItoPCI            Y                   -
SCentIf             N                   -
...
```

Query the current state of the krm module:

```
# kmsystem -q krm
Module              Configured          Loadable
```

```
================================================
krm                 Y                   Y
```

Set the LOADABLE flag to N:

```
# kmsystem -l N krm
# kmsystem -q krm
Module              Configured          Loadable
================================================
krm                 Y                   N

# cat /stand/system.d/krm
* @(#)$Revision: 1.1 $ $Date: 2000-04-18 11:18:16-06 $
* Kernel Resource Monitor system file
* (C) Copyright 1999 Hewlett-Packard Company
$VERSION        1
$CONFIGURE      Y
$LOADABLE       N
```

Refer to kmsystem(1M) man page for details.

kmtune(1M)

Displays/modifies/resets system tunables. kmtune simply modifies the system file.

```
# kmtune -q nproc
Parameter             Current Dyn Planned            Module     Version
================================================================================
nproc                    2068   -  2068

# kmtune -s nproc=3000
# kmtune -q nproc
Parameter             Current Dyn Planned            Module     Version
================================================================================
nproc                    2068   -  3000

# grep nproc /stand/system
nproc                3000

# kmtune -s nproc+1000
# kmtune -q nproc
Parameter             Current Dyn Planned            Module     Version
================================================================================
nproc                    2068   -  4000

# kmtune -r nproc
# kmtune -q nproc
Parameter             Current Dyn Planned            Module     Version
================================================================================
nproc                    2068   -  (20+8*MAXUSERS)
# grep nproc /stand/system
```

Refer to kmtune(1M) man page for details.

kmadmin(1M)

Loads/unloads a DLKM to/from the running kernel. Displays status information of modules.

```
kmadmin –L module_name ...| path_name ...
kmadmin –U module_name ...
kmadmin –u module_id ...
kmadmin –Q module_name ...
kmadmin –q module_id ...
```

```
kmadmin –S | -s
kmadmin –k
kmadmin –d directory_name | -D
```

- `-L` (*load*) Loads the module(s) to the running kernel. *`path_name`* is the complete path to the source code file (`mod.o`) of the module.
- `-U` (*unload*) Unloads the module(s) from the running kernel.
- `-u` (*unload*) Same as `–U` (needs module-ID as input)
  Note: `kmadmin –u 0` unloads all modules.
- `-Q` (*query*) Displays status information of the module.
- `-q` (*query*) Same as `-Q` (needs module-ID as input).
- `-S` (*status*) `kmadmin –Q` for all modules.
- `-s` (*status*) Short form of `kmadmin –S`.
- `-k` Prints a list of all statically configured modules.
- `-d` (*directory*) Specifies the search path to the modules. Default is `/stand/dlkm/mod.d/`
- `-D` (*directory*) Sets the search patch to the modules back to default: `/stand/dlkm/mod.d/`.

kmupdate(1M)

Update default kernel file and files associated with the kernel, or update specified kernel module(s).

(i) `kmupdate [kernel_file]`

(ii) `kmupdate –M module_name [[-M module_name]...] [-i|-a]`

form (i)

The specified kernel file will be moved to `/stand/vmunix` during next system **shutdown**. If no kernel file is specified, `/stand/build/vmunix_test` will be used. The associated kernel function set directory will be moved to `/stand/dlkm/`.

form (ii)

The files accociated with the specified module will be copied to their correct locations.

- `-i` (*immediately*) When specified, kmupdate will only attempt an immediate update.
- `-a` (*asynchronously*) When specified, kmupdate will update asynchronously without attempting an immediate update.

NOTE: `kmupdate` will first try `-i` option and (if this does not work) `-a` option.

Kernel Configuration changes at UX 11.11 (11i v1)

The only change compared to UX 11.00 is the introduction of **dynamic kernel tunables** (refer to Kernel Tunable section below).

Kernel Configuration changes at UX 11.22 (11i v1.6, Itanium)

UX 11.22 command line kernel configuration does not differ from UX 11.11.
The changes as of UX 11.22 are:

- As of UX 11.22 SAM's kernel configuration area has been replaced with a web GUI called kcweb.
- There are manual pages available for each kernel tunable.
- Several dynamic tunables have been added, some are obsolete (refer to Kernel Tunable section below).

kcweb

kcweb is a new web based interface, that offers the following features:

Kernel Configuration changes
You can modify tunables, add or remove static drivers, modify the state of kernel modules, compile a new kernel and reboot if needed.

Monitoring
Certain tunables can be monitored. A daily percatnage usage graph helps you to determine the appropriate adjustments to tunables.

Alarms
With kcweb alarms can be set for certain parameters. It is possible to be alarmed if e.g. nproc usage is greater than 80%. kcweb alarms use EMS notifications. All alarms are managed by kcalarm(1M).

Range checking
Another change compared to UX 11.00/11.11 is that the range checking of tunables is now performed directly in the kernel. kctune reports out of range errors, kcweb shows stderr output of kctune.

Help
You can access the man page of each tunable and get a detailed explanation.

Command preview
A great advantage compared to SAM is that for each task (e.g. modifying a tunable, module, or alarm) you can use the command preview feature by choosing the ? button; this will show the kernel configuration command invocation that will perform the requested task.

kcweb can be launched from SAM GUI (as of UX 11.23 additionally from SAM TUI) or manually. To access kcweb interface start the kcweb server on the system and access it from any browser.

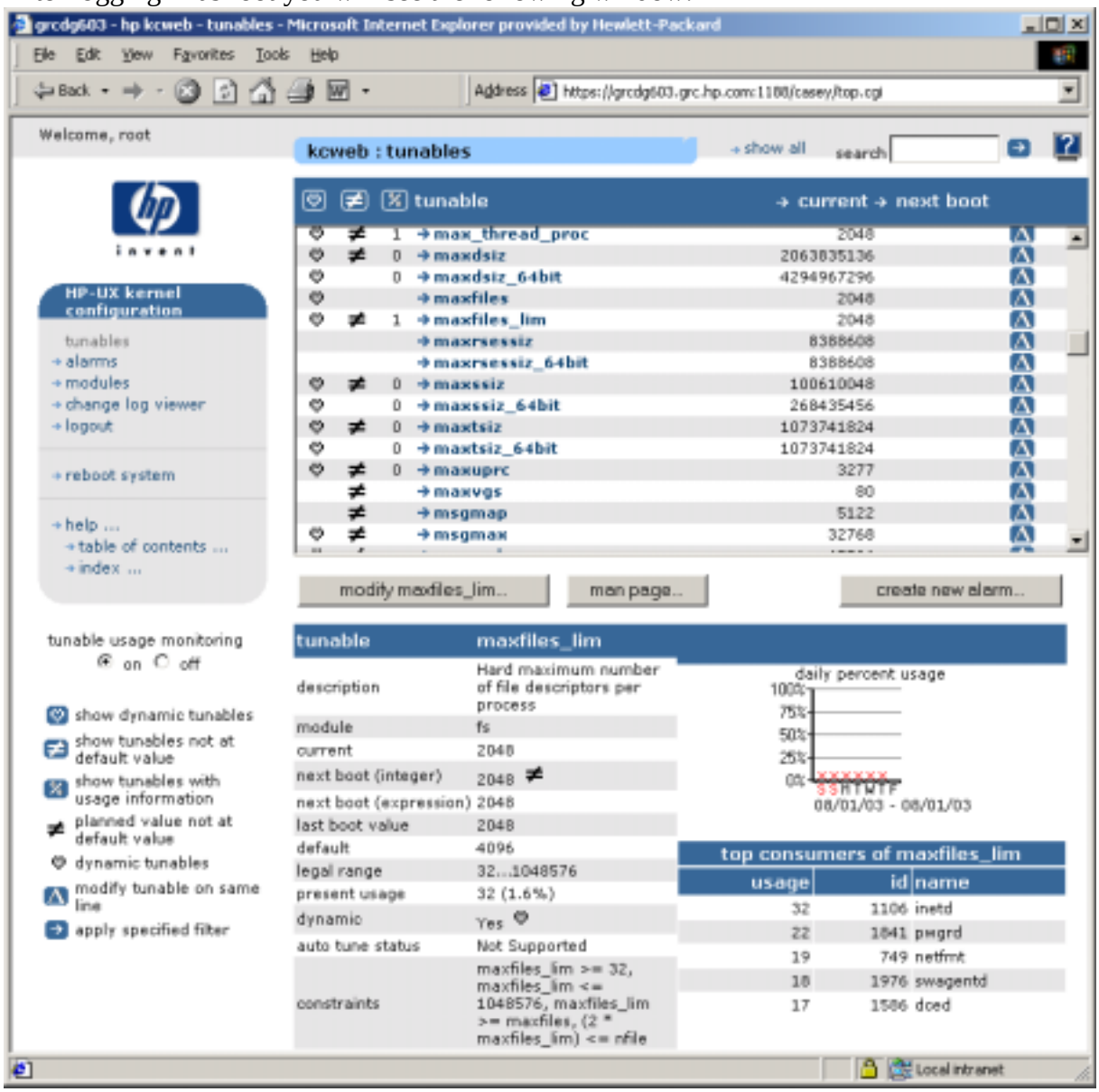
```
# kcweb -s startssl
Attempting to start server...
Server successfully started.
```

This creates server certificates (if needed), starts the kcweb administration server, connects to the server and presents a login screen.

At the following address you can access kcweb:

for UX 11.22: https://<hostname>:1188/cgi-bin/kcweb/top.cgi
for UX 11.23: https://<hostname>:1188/casey/top.cgi

After logging in as root you will see the following window:



To stop the web server do:

```
# kcweb -s stop
```

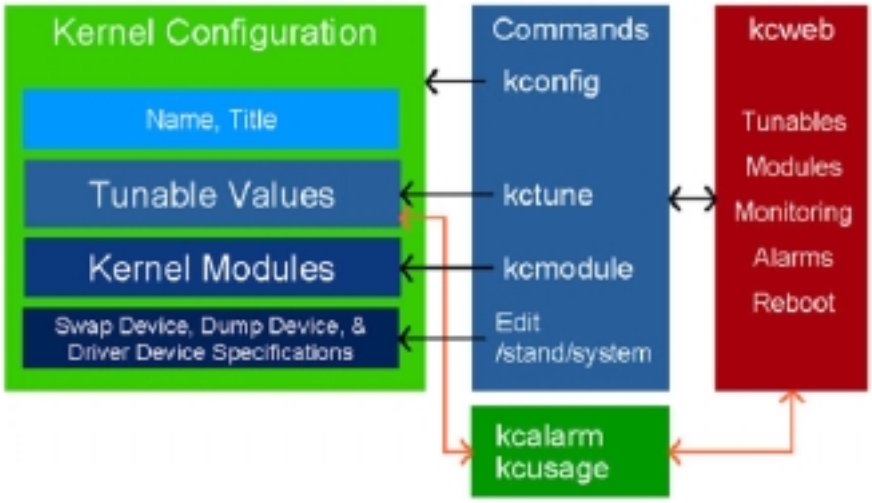
Kernel Configuration changes at UX 11.23 (11i v2, Itanium)

General changes

UX 11.23 introduces some new features. The primary design goals were:

- No longer compile the kernel and reboot
- No longer copy kernel executables
- Instead you manage *Kernel Configurations*

The main feature is the kernel configuration (KC) itself. Logically, a KC is a collection of all the administrators choices and settings needed to determine the behaviour and capabilities of the HP-UX kernel. A kernel configuration includes:



Physically, a KC is a directory under `/stand` that contains the files needed to realize the specified behaviour. The directory includes:

- An HP-UX kernel executable
- A set of HP-UX kernel module files
- A kernel registry database, containing all of the above settings
- A system file, describing the above settings in a human-readable form
- Various other implementation-specific files

The entire kernel configuration treated as a single unit that can be copied, backed up, selected by name, etc. You do not recompile the kernel anymore but modify a KC.

New commands

There are three new primary commands to manage kernel configurations, `kconfig(1M)`, `kctune(1M)` and `kcmodule(1M)`.

kconfig(1M)

is used to manage whole kernel configurations. It allows configurations to be saved, loaded, copied, renamed, deleted, exported, imported, etc. It can also list existing saved configurations and give details about them.

```
# kconfig -v
Configuration backup
Title         Automatic Backup
Save Time     Tue Jul 22 13:09:53 2003
Modify Time   Tue Jul 22 13:09:53 2003
Kernel Path   /stand/backup/vmunix

Configuration last_install
Title         Created by last OS install
Save Time     Thu Jul  3 09:23:03 2003
Modify Time   Thu Jul  3 09:23:04 2003
Kernel Path   /stand/last_install/vmunix
```

kctune(1M)

is used to manage kernel tunable parameters. kctune will display or change the value of any tunable parameter in the currently running configuration or any saved configuration.

```
# kctune nproc=2000
       * 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)      2000  2000

# kctune nproc=default
       * The automatic 'backup' configuration has been updated.
       * The requested changes have been applied to the currently
         running system.
Tunable            Value  Expression  Changes
nproc    (before)   2000  2000        Immed
         (now)      4200  Default
```

kcmodule(1M)

is used to manage kernel modules. Kernel modules can be device drivers, kernel subsystems, or other bodies of kernel code. Each module can be unused, statically bound into the main kernel executable, or dynamically loaded. kcmodule will display or change the state of any module in the currently running configuration or any saved configuration.

```
# kcmodule -v cdfs
Module              cdfs  (0.1.0)
Description         CD File System
State               auto (best state)
State at Next Boot  auto (best state)
Capable             auto static loaded unused
Depends On          interface HPUX_11_23:1.0

# kcmodule -v c8xx
Module              c8xx  [3EFFDFF0]
Description         SCSI C8xx Driver
State               static (best state)
State at Next Boot  static (best state)
Capable             static unused
```

```
Depends On          module wsio:0.0.0
                    interface HPUX_11_23:1.0

# kcmodule -v audio
Module              audio  [3EFFDFF0]
Description         Audio Driver
State               unused
State at Next Boot  unused
Capable             static unused
Depends On          module wsio:0.0.0
                    module beep:0.0.0
                    interface HPUX_11_23:1.0
```

kclog(1M)

is used to view and search the contents of the kernel configuration log file.

kcpath(1M)

is used by scripts to determine the path of the running kernel.

Deprecated/obsoleted commands

The commands `mk_kernel(1M)`, `kmtune(1M)` and `kmpath(1M)` present in previous HP-UX releases can still be used. They have been re-implemented as small shell scripts that invoke the new commands listed above. These commands will be removed in a future release.

`mk_kernel`	invokes `kconfig` (`-i`)
`system_prep`	invokes `kconfig` (`-e`)
`kmtune`	invokes `kctune`
`kmpath`	invokes `kcpath`
`config`	obsolete
`kmadmin`	obsolete
`kminstall`	obsolete
`kmmodreg`	obsolete
`kmsystem`	obsolete
`kmupdate`	obsolete

Overview of changes

UX 11.22 and before	UX 11.23
Use SAM to configure kernel	Use `kcweb` to configure the kernel
System files	Kernel configurations
Recompile and link the kernel	Reconfigure the kernel
Configure the kernel by editing `/stand/system` and run `mk_kernel`	KC commands (`kconfig`) automatically update the running kernel or relink it, if necessary
Always reboot after kernel changes	Use KC commands or kcweb to tune the kernel; changes will be dynamic, if possible.
Use `cp` to make a backup kernel executable	Use `kconfig -c` to copy a kernel configuration
Copy vmunix to another system	Use `kconfig` to import a kernel configuration
Use `system_prep` to create a current system file	System file is kept up to date automatically. Use `kconfig –e` to create a system file from a saved configuration
Manage DLKMs with `kminstall`, `kmsystem`,	Manage DLKMs using `kcmodule` or kcweb.

kmmodreg, kmupdate and config.	
Header, master and space files in /usr/conf	Metadata built into modules

Examples of changes

	Task	UX 11.00 - 11.22	UX 11.23
Tunables	List all tunables	kmtune (no options)	kctune
	List all tunables detailed	kmtune -l	kctune -v
	Query a tunable	kmtune -q maxuprc	kctune maxuprc
	Set a tunable to value	kmtune -u –s maxuprc=512	kctune maxuprc=512
	Increment tunable by value	kmtune -u -s maxuprc+128	kctune maxuprc+=128
	Set a value using a formula	kmtune -u -s maxuprc=nproc/30	kctune maxuprc=nproc/30
	Set tunable to default value	kmtune -r maxuprc	kctune maxuprc=default
	Set tunable to at least *n*	not possible	kctune maxuprc>=512
	Hold for next boot	kmtune –s maxuprc=512	kctune -h maxuprc=512
	List values held for next boot	kmtune -d	kctune –D
	Set a tunable in a saved configuration	not possible	kctune –c WeekendConfig maxuprx=512
	Create a user defined tunable	not possible	kctune –u _mytunable=256
Modules	Load a module	kmadmin –L *module*	kcmodule *module*=loaded
	Unload a module	kmadmin –U *module*	kcmodule *module*=unused
	Query a module	kmadmin –Q *module*	kcmodule -v *module*
	Print status of all modules	kmadmin -s	kcmodule
	Print detailed status of all modules	kmadmin -S	kcmodule -v
	Print all static modules	kmadmin -k	kcmodule
	Add, delete or update module	kminstall	no longer needed
	(Un-)register module with the running kernel	kmmodreg	no longer needed
	List all modules	kmsystem (no options)	kmcmodule
	Set module to configured and loadable.	kmsystem -c y -l y *module*	kcmodule *module*=loaded
	Set module to configured and not loadable.	kmsystem -c y -l n *module*	kcmodule *module*=static
	Set module to unconfigured.	kmsystem -c n *module*	kcmodule *module*=unused
	Query module state	kmsystem -q *module*	kcmodule -v *module*
	Configure a loadable module	mk_kernel -M	no longer needed
	Path to running kernel (/stand/vmunix)	kmpath (no options)	kcpath -x
	Name of running kernel (vmunix)	kmpath -k	kcpath -b
	Path to current kernel function set directory (/stand/dlkm)	kmpath -c	kcpath -d
	Mark kernel/configuration for next boot	kmupdate *kernel*	kconfig -n *configuration*

Procedure to modify the kernel configuration

If you like to modify only a few tunables or modules use kctune or kcmodule respectively:

```
# kctune nproc=1000
# kcmodule krm=loaded
```

If tunables/modules are dynamic you're done, else you need to reboot.

In order to perform many changes at once, e.g. apply tunable settings from another system do the following:

1) Export the current kernel configuration to a file:

```
# kconfig -e system_config
       * The current configuration (including any changes being held for
         next boot) has been exported to /root/system_config.
```

NOTE: The obsolete `system_prep` script has been rewritten to incoke `kconfig -e`

2) Now change the tunable values by editing this file:

```
# vi system_config
*
* Created on Thu Jul 31 11:08:37 2003
*
version 1
configuration nextboot "" [3f27e819]
*
* Module entries
*
module drmfglrx auto 0.1.0
module drmfgl auto 0.1.0
module gvid_him_rad auto 0.1.0
...
...
module lba best [3F0E6070]
module sba best [3F0E6070]
module root best [3F0E6070]
*
* Swap entries
*
*
* Dump entries
*
*
* Driver binding entries
*
*
* Tunables entries
*
tunable secure_sid_scripts 0x0
tunable nstrpty 60
tunable dbc_max_pct 20
tunable vx_ninode 30000
```

NOTE: Do not try to change modules. Tunables only.

3) Now import the modified configuration:

If you modified only dynamic tunables, then changes will be applied immediately:

```
# kconfig -i 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.
       * system has been imported.  The changes have been applied to the
```

```
          currently running system.
```

If you modified at least one static tunable it takes a reboot to apply the changes:

```
# kconfig -i system_config
       * The automatic 'backup' configuration has been updated.
NOTE:    The configuration being loaded contains changes that cannot be
         applied immediately:
      -- The tunable vx_ninode cannot be changed in a dynamic fashion.
NOTE:    The changes will be held for next boot.
       * system_config has been imported.  The changes will take effect
         at next boot.
```

Booting a kernel configuration

At the Boot Loader prompt enter:

`HPUX> boot` *`configname`*

Boot to failsafe mode (i.e. single user mode, hard coded tunables, no DLKMs):

```
HPUX> boot –tm [configname]
```

kcweb changes

The URL used to access the kcweb interface has changed to https://<hostname>:1188/casey/top.cgi

UX 11.23 introduces the `waconf(1M)` command which allows to configures automatic startup of the web administration server used by kcweb (and pdweb) at boot time. `waconf` configures autostart by editing `/etc/inetd.conf` and `/etc/services`.

You can turn autostart on, off or query the current state using:

```
# waconf [-a on|off]
```

Example:

```
# waconf -a on
/etc/inetd.conf and /etc/services have been edited to allow
web administration tools to be autostarted using port 1110.

If NIS is in use you may need to edit /etc/nsswitch.conf or
change the configuration of the NIS server for the new webadmstart
service to work.

Network firewalls can also keep autostarting from working.

# waconf
Autostart is enabled

# grep web /etc/services
webadmstart        1110/tcp                # start web admin server
```

Errors during Kernel Compilation (not UX 11.23)

NOTE: This section does not apply to UX 11.23 since the kernel is no longer compiled there.

You have probably seen `mk_kernel` failing with the error message *unsatisfied symbols*:

```
/stand/build/ # mk_kernel

NOTE:    Building a new kernel based on template file "/stand/system"
Loading the kernel...
/usr/ccs/bin/ld: Unsatisfied symbols:
   kpageoutcnt (data)
/usr/ccs/bin/ld: (Warning) Linker features were used that may not be
supported in future releases. The +vallcompatwarnings option can be
used to display more details, and the ld(1) man page contains
additional information. This warning can be suppressed with the
+vnocompatwarnings option.
*** Error exit code 1

Stop.
         make failure.
```

NOTE: The warning can be gracefully ignored.

In most of the cases this happens after installation of a patch bundle. Most od the time, it is because the dependencies within this patch bundle are not properly resolved. But lets check what is going on.
The problem begins with the `ld` command. If you go back, you will see that the `make` command uses `ld` to link objects. This is why you get a `make` failure. The `ld` command was attempting to satisfy the symbol table to generate the `vmunix_test` kernel file. This command failed, because it could not satisfy some of the external symbols or .o files.

In this example, "`kpageoutcnt`" is the unsatisfied symbol that needs to be resolved. An unsatisfied symbol may be tracked down by using the following steps.

1) Finding the object file

In order to find the library including the object file where the symbol is defined, you may use the *HP-UX Kernel Patch Symbol Query* tool: http://dumpy.grc.hp.com/kernelsym.html (HP internal) in case you have access to the HP intranet.Given the symbol, it reports the object file and the patch(es) that include this object file.

Otherwise you need a functioning system (properly patched and as close to the system in question) because the symbol is missing on the system where mk_kernel fails.

On the functioning system do:

```
# nm -A /usr/conf/lib/lib* | grep kpageoutcnt
[60]     |       28|       4|OBJT |GLOB |0|  .sdata|/usr/conf/lib/libhp-
ux.a[vm_vhand.o]:kpageoutcnt
[42]     |        0|       0|OBJT |GLOB |0|   UNDEF|/usr/conf/lib/libhp-
ux.a[vm_swp.o]:kpageoutcnt
```

```
[118]    |        0|       0|OBJT |GLOB |0|   UNDEF|/usr/conf/lib/libhp-
ux.a[vfs_bio.o]:kpageoutcnt
[111]    |        0|       0|OBJT |GLOB |0|
UNDEF|/usr/conf/lib/libnfsnew.a[nfs_vnops.o]:kpageoutcnt
[87]     |        0|       0|OBJT |GLOB |0|
UNDEF|/usr/conf/lib/libnfsnew.a[nfs_subr.o]:kpageoutcnt
[141]    |        0|       0|OBJT |GLOB |0|
UNDEF|/usr/conf/lib/libnfsnew.a[nfs3_vnops.o]:kpageoutcnt
```

The first line gives the object file where the symbol is defined. The other objects are just referencing it which is shown by the keyword UNDEF. So now you know the object is vm_vhand.o and the library that contains the object is libhp-ux.a

NOTE: At 32 bit systems the nm output looks a little bit different:

```
# nm -A /usr/conf/lib/lib* | grep kpageoutcnt
/usr/conf/lib/libhp-ux.a:kpageoutcnt|1073746084|extern|data
|$SHORTDATA$
/usr/conf/lib/libhp-ux.a:kpageoutcnt|          |undef |data   |
/usr/conf/lib/libhp-ux.a:kpageoutcnt|          |undef |data   |
```

Since the object file (.o) is not returned you need to search the library in order to find the object file:

```
# nm /usr/conf/lib/libhp-ux.a | grep -e "Symbols from" -e kpageoutcnt
| more
Symbols from /usr/conf/lib/libhp-ux.a[vm_vhand.o]:
kpageoutcnt         |1073746084|extern|data   |$SHORTDATA$
Symbols from /usr/conf/lib/libhp-ux.a[vm_vfd.o]:
Symbols from /usr/conf/lib/libhp-ux.a[vm_vdma.o]:
Symbols from /usr/conf/lib/libhp-ux.a[vm_vas.o]:
Symbols from /usr/conf/lib/libhp-ux.a[vm_unhash.o]:
Symbols from /usr/conf/lib/libhp-ux.a[vm_uio.o]:
Symbols from /usr/conf/lib/libhp-ux.a[vm_textcache.o]:
Symbols from /usr/conf/lib/libhp-ux.a[vm_text.o]:
Symbols from /usr/conf/lib/libhp-ux.a[vm_swp.o]:
kpageoutcnt         |          |undef |data   |
Symbols from /usr/conf/lib/libhp-ux.a[vm_swalloc.o]:
Symbols from /usr/conf/lib/libhp-ux.a[vm_sw.o]:
...
...
```

so vm_vhand.o is the object file.

2) Checking the installed version of the object

Let see which version of `vm_vhand.o` is currently installed. On the bad system, do:

```
# what /usr/conf/lib/libhp-ux.a | grep vm_vhand
     vm_vhand.c $Date: 1999/02/24 09:46:41 $Revision: r11ros/3 PATCH_11.00
(PHKL_17869)
```

A `what` command on the kernel (`/stand/vmunix`) should give the same result.

PHKL_17869 is the patch where the object file was last modified. Most likely a successor of this patch is installed.

3) Determine the patches that affect the object

Now we need to find any existing patch that affects the object file `vm_vhand.o` .If you have access to the HP intranet you may use the *HP-UX Kernel Patch Symbol Query* tool again or another tool that allows you to find a patch by file or object:
http://wtec.cup.hp.com/~patches/forms/patchfile.html (HP internal).

The input

Release: UX 11.00
File Names: vm_vhand

yields the results

```
/usr/conf/lib/libhp-ux.a(vm_vhand.o)  --> PHKL_18543 - 11.00
PM/VM/UFS/async/scsi/io/DMAPI/JFS/perf patch

/usr/conf/lib/libhp-ux.a(vm_vhand.o)  --> PHKL_25906 - 11.00
Probe,IDDS,PM,VM,PA-8700,asyncio,T600,FS
```

These are the latest versions of patches that affect vm_vhand.o. Now, we have two patches. Most likely, the newer one (PHKL_25906) or one of its predecessors introduced the symbol kpageoutcnt. A reinstallation of PHKL_18543 (where the symbol was not yet defined) would have overwritten `vm_vhand.o` and therefore the definition of `kpageoutcnt`.

Checking the patch catalog http://wtec.cup.hp.com/~patches/catalog/ (HP internal) shows

- PHKL_18543 is the latest successor of PHKL_17869.
- version of `vm_vhand.o` from PHKL_18543 is
  ```
  /usr/conf/lib/libhp-ux.a(vm_vhand.o):
        vm_vhand.c $Date: 1999/02/24 09:46:41 $Revision: r11ros/3
  PATCH_11.00 (PHKL_17869)
  ```
- version of `vm_vhand.o` from PHKL_25906 is:
  ```
  /usr/conf/lib/libhp-ux.a(vm_vhand.o):
        vm_vhand.c $Date: 2000/04/12 03:18:15 $Revision: r11ros/6
  PATCH_11.00 (PHKL_21532)
  ```

NOTE: There are two versions on any 64 bit system. Be sure to pick the right one. To check if this is a 32 bit or 64 bit system, run `getconf KERNEL_BITS`.

4) Comparing with swlist of bad system

Checking swlist output of the bad system shows that PHKL_25525, a predecessor of PHKL_25906 is installed.
This contradicts to the fact that `vm_vhand` version on the system is from PHKL_17869!
In this case, PHKL_25525 had been overwritten by PHKL_18543.

5) Reinstalling the patch(es)

The solution would be to reinstall this patch or one of it's successors without the dependencies.

NOTE: In this case, a line in the sand patch (PHKL_16751/PHKL_16750 for UX 10.20 or PHKL_18543 for UX 11.00) had been reinstalled which is not permitted. See Patches Chapter for an explanation of what line in the sand patches are. Running the check_patches utility (see Patches Chapter) gives you the object files that have been overwritten by a line in the sand patch. If there are such files, you need to reinstall all the patches that check_patches warns about (excluding the line in the sand patch).

Kernel Tunables

Tunable Types

There are 5 different **types** of kernel tunables:

Type	Description	Introduced
Static	Traditional tunable parameters	prior to UX 10.20
Dynamic	Can be changed without rebooting the system	in UX 11.11
Automatic	If an atutomatic tunable is set to default then the system determines an appropriate value at boot time. Automatic tunables are not changed automatically while the system is running	in UX 11.23
Private	Tunables hidden from listings, but can be managed by commands	in UX 11.22
User-Defined	Used like variables in other tunable formulas	in UX 11.23

The following tunables are **dynamic**, i.e. modification does not need a reboot:

dynamic as of 11.11	additionally as of 11.22	additionally as of 11.23
core_addshmem_read	executable_stack	aio_* tunables (7)
core_addshmem_write	ksi_alloc_max	alwaysdump
maxfiles_lim	max_acct_file_size	dbc_max_pct
maxtsiz	max_thread_proc	dbc_min_pct
maxtsiz_64bit	maxdsiz	dontdump
maxuprc	maxdsiz_64bit	dump_compress_on (new)
msgmax	maxssiz	enable_idds
msgmnb	maxssiz_64bit	fs_symlinks
scsi_max_qdepth	nkthread	ncdnode
semmsl	nproc	st_* tunables (3)
shmmax	physical_io_buffers	vxfs_ifree_timelag (new)
shmseg	secure_sid_scripts	
	shmmni	

The following tunables are **automatic**, i.e. are not modified by the user anymore:

automatic as of 11.11	additionally as of 11.22	additionally as of 11.23
	maxswapchunks	nfile
	ncallout	nflocks
	ncallout	physical_io_buffers
		maxfiles

The following tunables are **obsoleted**:

obsoleted in 11.11	obvsolteted in 11.22	obsoleted in 11.23
vx_noifree	bootspinlocks	maxusers
	clicreservedmem	shmem
	ndilbuffers	sema
	nni	mesg
	netisr_priority	
	netmemmax	
	spread_UP_drivers	

How to identify the tunable type

Dynamic tunables are marked `Immed` in the kctune listing because their value can be changed immediately.
Automatic tunables whose values are automatically tuned by the system at boot time are marked `Auto`.

- **Static** tunable example:

  UX 11.11, 11.22:

```
# kmtune -q maxvgs
Parameter             Current Dyn Planned                    Module     Version
================================================================================
maxvgs                      10   -  10
```

  UX 11.23:

```
# kctune | grep maxvgs
maxvgs                         10  Default

# kctune -v maxvgs
Tunable             maxvgs
Description         Maximum number of LVM volume groups
Module              lvm
Current Value       10 [Default]
Value at Next Boot  10 [Default]
Value at Last Boot  10
Default Value       10
Constraints         maxvgs >= 1
                    maxvgs <= 256
Can Change          At Next Boot Only
```

- **Dynamic** tunable example:

  UX 11.11, 11.22:

```
# kmtune -q shmmax
Parameter             Current Dyn Planned                    Module     Version
================================================================================
shmmax            10000000000  Y  10000000000
```

  UX 11.23:

```
# kctune shmmax
Tunable       Value  Expression  Changes
shmmax   1073741824  Default     Immed

# kctune -v shmmax
Tunable             shmmax
Description         Maximum size of a shared memory segment (bytes)
Module              sysv_shm
Current Value       1073741824 [Default]
```

```
Value at Next Boot  1073741824 [Default]
Value at Last Boot  1073741824
Default Value       1073741824
Constraints         shmmax >= 2048
                    shmmax <= 4398046511104
Can Change          Immediately or at Next Boot
```

- **Automatic** tunable example (UX 11.23 only):

```
# kctune | grep nfile
nfile                       65536  Default     Auto

# kctune -v nfile
Tunable             nfile
Description         Maximum number of file descriptors (system-wide)
Module              fs
Current Value       65536 [Default]
Value at Next Boot  16384 [Default]
Value at Last Boot  65536
Default Value       65536
Constraints         nfile >= 2048
                    nfile <= 2147483647
                    nfile >= (2 * maxfiles_lim)
Can Change          Automatic Tuning Enabled
```

NOTE: As of UX 11.22 there are manual pages available for each tunable.

Tunable Usage

Kernel tunables limit the use of system resources. It may be interesting to find out the current usage of system wide kernel parameters. A typical question would be:

How many processes are currently running and how many processes am i allowed to start before the system wide limitation defined by the kernel tunable nproc is violated?

As of UX 11.22 you can use kcweb in order to figure this out. In any case you can use the crash dump debugger Q4 as well as P4 are capable of doing this. Q4 is included in the HP-UX Core-OS. Refer to the Dump Chapter in order to get the most recent version of Q4.

For Q4 you may need to preprocess the kernel for debugging first:

```
# /usr/contrib/bin/q4pxdb /stand/vmunix
q4pxdb64: /stand/vmunix is already preprocessed
PXDB aborted.
```

Now start Q4 using the running kernel `/stand/vmunix` and the device file to access the memory /dev/mem

```
# /usr/contrib/bin/q4 /stand/vmunix /dev/mem
@(#) q4 $Revision: A.11.11c $ $Thu Jan 25 18:05:11 PDT 2001 0
...
...
```

At the q4 prompt, you can print the current setting of a tunable, e.g.

```
q4> nproc
05670   3000    0xbb8
```

For P4 do:

```
p4> Let nproc
05670   3000    0xbb8
```
or
```
p4>dec nproc
3000
```

In order to see the current usage you need to load the structure (e.g the proc table) and keep all non-empty entries. This is done differently for each tunable depending on the underlying kernel structure.

Here are some examples:

- nproc - maximum number of processes system-wide

```
q4> load struct proc from proc max nproc; keep p_stat
loaded 3000 struct proc's as an array (stopped by max count)
kept 545 of 3000 struct proc's, discarded 2455
```

This means 545 out of 3000 (=nproc) processes are currently existing.

NOTE (UX 11.11 and beyond):

As of UX 11.11, the static proc table has been replaced with a dynamically linked list:

```
q4> nproc
03720   2000    0x7d0

q4> load struct proc from proc_list next p_factp max nproc
loaded 272 struct procs as a linked list (stopped by null pointer)
```

this means 272 out of 2000 (=nproc) processes are currently existing.

- nkthread - maximum number of processes system-wide

```
q4> load struct kthread from kthread max nkthread; keep kt_stat
loaded 532 struct kthread's as an array (stopped by max count)
kept 130 of 532 struct kthread's, discarded 402
```

This means 130 out of 532 (=nkthread) threads are currently existing.

NOTE (UX 11.11 and beyond):

As of UX 11.11, the static proc table has been replaced with a dynamically linked list:

```
q4> nkthread
07020   3600    0xe10

q4> load struct kthread from kthread_list next kt_factp max nkthread
loaded 367 struct kthreads as a linked list (stopped by null pointer)
```

this means 367 out of 3600 (=nkthread) threads are currently existing.

- nfile - maximum number of open files (file descriptors) system-wide.

```
q4> load struct file from file max nfile; keep f_count
```

```
loaded 5010 struct files as an array (stopped by max count)
kept 740 of 5010 struct file's, discarded 4270
```

This means 740 out of 5010 (=nfile) files are currently open.

- nflocks - maximum number of file locks system-wide (is automatic as of UX 11.23)

```
q4> load struct locklist from &locklist max nflocks; keep ll_count
loaded 200 struct locklist's as an array (stopped by max count)
kept 37 of 200 struct locklist's, discarded 163
```

This means 30 out of 200 (=nflocks) file locks are currently in use.

- ninode - maximum number of HFS inodes system-wide

```
q4> load struct inode from inode max ninode; keep i_number
loaded 1000 struct inodes as an array (stopped by max count)
kept 23 of 1000 struct inode's, discarded 977
```

This means 23 out of 1000 (=ninode) HFS inodes are currently in use.

- npty - maximum number of pseudo-tty data structures system-wide.

```
q4> load struct pty_info from pty_info max npty; keep pty_state
loaded 200 struct pty_info's as an array (stopped by max count)
kept 3 of 200 struct pty_info's, discarded 197
```

This means 3 out of 200 (=npty) PTYs are currently in use.

- nstrpty - maximum number of streams-based PTYs system-wide

```
q4> load struct pty_s from &str_pty max nstrpty; keep pt_alloc
loaded 60 struct pty_ss as an array (stopped by max count)
kept 6 of 60 struct pty_s's, discarded 54
```

This means 6 out of 60 (=nstrpty) streams-based PTYs are currently in use.

- ncallout - maximum number of kernel timeouts

```
q4> load struct callout from callout max ncallout; keep c_flag !=
FREE_CALLOUT
loaded 548 struct callout's as an array (stopped by max count)
kept 77 of 548 struct callout's, discarded 471
```

This means 77 out of 548 (=ncallout) kernel timouts are currently scheduled.

NOTE (UX 11.11 and beyond):

As of UX 11.11, callouts scheduled are per cpu which makes it too complicated to determine the overall usage. There is a perl script that does it:

```
q4> include callout.pl
loading /usr/contrib/Q4/lib/q4lib/callout.pl ...
```

```
 run Callout; executes the callout debugging script
 Creates global summary file "callout_summary" as well
 as per-spu files "callout_spu<n>"

q4> run Callout
Processing spu 0 ...
...
```

NOTE: For P4 simply replace the “&” operator with the “@” operator:

```
q4> load struct locklist from @locklist max nflocks; keep ll_count
Loaded  200 entries in 'DefaultView'
Kept     37 entries in DefaultView
```

As of UX 11.22 there is kcweb to monitor tunable usage statistics.

Additional information

Detailed explanations of the kernel tunables including their default, minimum and maximum values can be found on the public HP document server:
http://docs.hp.com/hpux/onlinedocs/TKP-90202/TKP-90202.html

DLKMs are described in the HP-UX 11.00 release notes:
http://docs.hp.com/hpux/onlinedocs/B3782-90716/B3782-90716.html

UX 11.23 kernel configuration whitepaper:
http://www.hp.com/products1/itanium/infolibrary/whitepapers