

Chapter 10

-

Crash Dumps

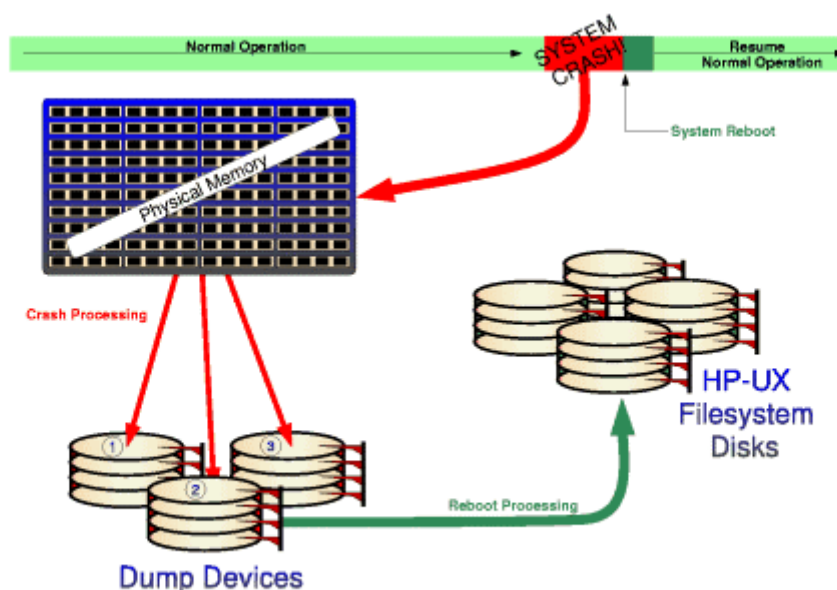
INDEX

A little bit of theory	3
Crash events	3
What happens when a system crashes?.....	4
How to configure dump devices	8
Choosing dump devices	8
Configuration steps	9
The dump/savecrash process	11
Writing the memory image to the dump devices	11
Saving the dump to the filesystem.....	14
Analysis of the dump	16
About the crashinfo utility	16
How to obtain and install and execute crashinfo	16
Analysis beyond standard crashinfo output	18
crashinfo output example.....	24
Patches related to crash dumps	30
Additional information	32

Did you ever experience a system that was hung or crashed unexpectedly? This chapter explains how to configure a system for crash dump, how to install dump analysis tools and how to use them in order to quickly isolate the cause of the problem.

A little bit of theory

When the system crashes, HP-UX tries to save the image of physical memory (*core*), or certain portions of it, to predefined locations called *dump devices*. Then, during the following reboot, a special utility (*savecrash*) is invoked from a rc-script that copies the memory image together with the current kernel from the dump devices to the file system. Once there, you can analyze the memory image with a debugger. The following picture shows the action flow:



Crash events

An abnormal system reboot is called a *crash*. There are many reasons that can cause a system to crash; hardware malfunctions, software panics or even power failures. On a properly configured system, these will typically result in a crash dump being saved. The operating system logs a crash event for each reason that triggers a crash. There is usually one crash event per-processor. Although it is not uncommon to see two or more crash events associated with the same processor.

There are three different types of crash events: **PANIC**, **TOC** and **HPMC**:

PANIC

The crash event type *panic* refers to crashes initiated by the HP-UX operating system (software crash event). We differentiate between *direct* and *indirect* panics.

A *direct panic* refers to a subsystem calling directly the `panic()` kernel routine upon detection of an unrecoverable inconsistency, for example:

- panic ("wait_for_lock: Already own this lock!");
- panic ("m_free: freeing free mbuf");
- panic ("virtual_fault: on DBD_NONE page");
- panic ("kalloc: out of kernel virtual space");

An *indirect panic* refers to a crash event as a result of trap interruption which could not be handled by the operating system. For example when the kernel accesses a non-valid address, a Data page fault (trap type 15) would result. The trap handler will save some state information and then call the panic() routine to bring the box down in an orderly manner. This is indirect since panic() is called at a point slightly later than the trap condition that caused the failure. Some examples

- trap type 15, Data page fault
- trap type 18, Data memory protection fault
- trap type 6, Instruction page fault

TOC

The crash event type *TOC* refers to crashes initiated by a *Transfer-Of-Control* sequence. There are three different ways of getting a TOC event for a CPU:

- Operator initiated TOC (eg, manually pushing a TOC button, or cycling the power button 3 times on some systems, or using the TC command in console mode).
- MC/ServiceGuard initiated TOC (eg, when it is unable to maintain contact with the cluster daemon).
- Crash path initiated TOC. On multi-processors systems, the processor taking the initial crash event (eg, a panic) will cause the other processors to perform TOC automatically.

A manual TOC is usually done when a system is hung or unresponsive. This way the crash dump can be analysed to determine root cause.

HPMC

The HPMC crash event type refers to High Priority Machine Check crashes initiated by the hardware due to hardware inconsistencies or malfunctions such as a Data Cache parity error. Getting an HPMC does not always mean that the hardware is at fault. The HPMC tombstone needs to be analyzed to determine if the hardware was really at fault. Software defects can result in HPMC crash events, but are typically very rare in production quality software.

NOTE: on Itanium (IA-64) systems the naming is slightly different:

HPMC = MCA (Machine Check Abort)
TOC = INIT

What happens when a system crashes?

Now that you understand the different types of crash events (panic, toc and hpmc), let's see what the system does to process these events. Processing these events usually requires an

interaction between the hardware and operating system software. There are well defined architected interfaces between hardware and software. For example, PDC entry points (processor firmware) on the processors and Interruption Vector Table (IVA) in the kernel. These interfaces allows the hardware to trigger software entry points to initiate logging, analysis and error recovery to be performed after a hardware fault or vice versa.

Some of the information presented here may be quite indepth on first reading. You may skim through them initially. It is important to grasp the concept presented here since any investigative dump analysis work begins with the crash events. It is worthwhile understanding what the system does in response to crash events and what crucial pieces of information are saved and where they are stored.

We categorise the crash events into two classes; hardware crash events and software crash events. Here is a description of what the system does to process these.

Hardware crash events

A hardware crash event can be *High Priority Machine Check (HPMC)*, *Low Priority Machine Check (LPMC)* or *Transfer of Control (TOC)*. The machine checks are typically caused by hardware malfunctions or certain classes of bus errors. TOC on the other hand is usually initiated by the operator in response to system software being stuck in an error state.

When a hardware crash event occurs, the processor immediately branch to PDC entry point; PDCE_CHECK for HPMC and LPMC faults, and PDCE_TOC for TOC. The implementation details of these PDC entry points are processor dependant. Fundamentally they save the processor's state (general, control, space and interruption registers) into *Processor Internal Memory (PIM)*. The processor then vectors back into the operating system entry points; HPMC_Vector or TOC_Vector. These entry points are defined in the IVA (*Interruption Vector Table*) and MEM_TOC in Page Zero respectively.

On entry into the kernel, a crash event entry is created. The operating system makes a pdc call (PDC_PIM) to read the processor's state information from PIM into a *Restart Parameter Block (RPB)*. As such the RPB structure contains information pertinent to the understanding of the crash. For example, the *Program Counter (PC)* in the RPB would indicate what routine was executing at the time of HPMC/TOC event. Once the state has been saved, the operating system continues to dump physical memory to the dump device.

Software crash events

A software crash event occurs when *panic()* routine is called. This can either be a direct or indirect panics. For a software crash event, the PDC and PIM are not involved at all. As such, the first thing that panic() routine does is to save the processor state into the RPB structure. The panicking processor will also initiate a TOC to other processors, causing them to stop what they are doing closer to the point where the problem is detected. This is important to allow the cause of the panic to be identified.

panic() actually calls a leaf routine panic_save_register_state() to save the processor registers state. So the return pointer (rp) in the RPB structure actually points to the panic() routine. The instruction address (pcoq) is zeroed out in the RPB to prevent unwinding beyond panic since this is the point of interest. Since panic_save_register_state() is a leaf routine, the stack pointer (sp) in the RPB will be the same as that of panic().

For a *direct panic*, the RPB contains the processor's registers state of the routine which called

panic(). In other words, the RPB contains information closest to the point of failure and in the same context as the routine was called. Thus dump analysis begins with the RPB for direct panics.

For an *indirect panic*, the RPB contains the context of a trap handler and it does not reflect the value of the registers at the time of the fault. Please see the following diagram. An indirect panic is usually the result of a trap condition which cannot be resolved by the operating system. The trap handler needs to save the processor state information before bringing down the system gracefully with a panic call. The trap handler stores these registers state into a `save_state` structure. So for an indirect panic, the `save_state` structure contains information closest to the point of failure which triggered the trap condition. Thus dump analysis begins with the `save_state` for indirect panics.

After `panic()` has saved the state, it proceeds to dump physical memory to dump device.

PIM Tombstone

The Process Internal Memory or PIM is a storage area in a processor that is set at the time of an HPMC, LPMC, Soft Boot, or TOC, and is composed of the architected state save error parameters, and HVERSION-dependent (ie, processor dependent) regions. The internal structure of PIM is processor dependant. The `PDC_PIM` procedure is used to access PIM data.

Different systems have different methods of accessing PIM information. On some systems, there is a `pdinfo` program that allows online retrieval of this PIM data. This can be helpful to retrieve HPMC tombstone data for analysis. The script in `/sbin/init.d/pdinfo` automatically runs `pdinfo` command when HP-UX is booted and saves any tombstones in a file in the directory `/var/tombstones`. Up to 100 files can be saved. The file "ts99" is the most current, "ts98" is the next most current...."ts0" would be the oldest.

From a dump analysis point of view (especially HPMC/TOC), the RPB structure should be a reflection of the registers state in PIM since the information was copied from it. There are rare times when `rpb` values may not seem 'right'. If this is the case then it is better to use the register values in the PIM data as starting point for analysis. Some interesting registers are:

gr02	Return Pointer (rp)
gr30	Stack Pointer (sp)
cr17	Interruption Instruction Address Space Queue (pcsq)
cr18	Interruption Instruction Address Offset Queue (pcoq)
cr19	Interruption Instruction Register (iir)
cr20	Interruption Space Register (isr)
cr21	Interruption Offset Register (ior)
cr22	Interruption Processor Status Word (ipsw)
cr23	External Interrupt Request Register (eirr)
cr15	External Interrupt Enable Mask (eirr)

Save state structure

The `save_state` structure is used by the interrupt (`ihandler`) and trap (`thandler`) handlers to temporarily store away processor state (general, control, space and interruption registers) so that these handlers can safely reuse the registers. It will also allow the handlers to return to the point of interruption by restoring these register values from the `save_state`. The `save_state` structure (together with a frame marker) is typically allocated on the Interrupt Control Stack (ICS) or kernel stack.

Most of the processor registers are saved. However, some registers are not saved because they are irrelevant when returning to the point of interruption. Since these interrupt and trap handlers are executed frequently, it is crucial for performance reasons to save only what is necessary.

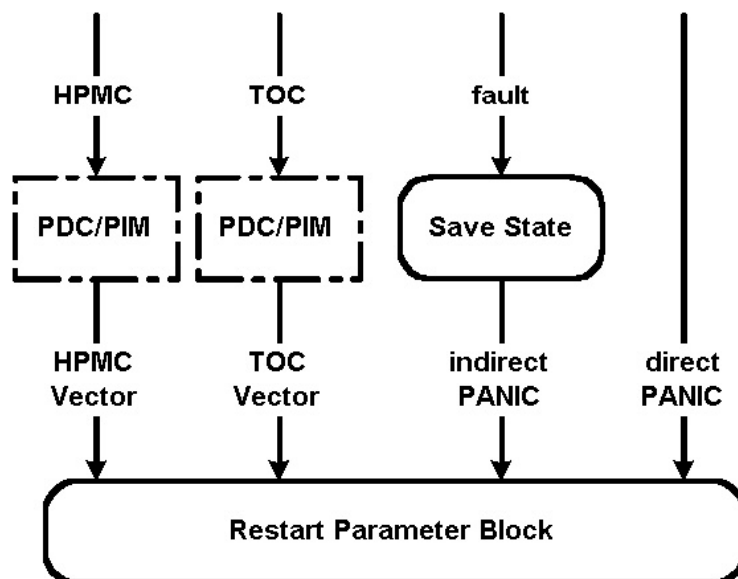
RPB structure

Every crash event will create a corresponding RPB structure to contain the processor state at the time of HPMC, TOC or panic. This register state allows us to understand what is happening at that point in time as well as provides a starting point for the stack unwind. The rpb structures are stored in a pre-allocated area in kernel static data area.

Unlike the `save_state` structure, the `rpb` structure will contain a more complete save of all the processor registers. For example, the `cr16` interval timer is saved in the `rpb` but not in the `save_state` structure. We can afford to save more registers in `rpb` since it is created during the crash path which is not a performance sensitive code path.

Crash event flowchart

Here is a diagram summarizing the above:



How to configure dump devices

In order to understand the following text you should be familiar with the basic concept of the Logical Volume Manager LVM. I make use of the these abbreviations:

VG = Volume Group
LV = Logical Volume
PV = Physical Volume

Choosing dump devices

Dump devices are volumes on the disk that are used to hold the entire memory image when the system crashes. The cumulative size of all specified dump devices has to be some MB larger than the amount of memory in order to hold the entire core. To determine the current size of physical memory:

```
# dmesg | grep Physical
Physical:524288 KB ,lockable:386672 KB ,available:454144 KB
```

As of UX 11.00 you can use `crashconf(1M)`:

```
# crashconf | grep Total
Total pages on system:      131072
Total pages included in dump: 30832
```

(A page is always 4KB)

NOTE: Increasing the amount of dumpspace is an important thing to do when adding more physical memory to the system.

Formerly the maximum size of a dump device was 2GB or more precise: the dump LV had to be placed within the first 2GB of the PV whereas newer systems support dump devices up to 4GB or since UX 11.00 even greater than 4GB.

It's important to mention that it's the Interface Card, not the disk, that defines whether the disk can be used for more than 4GB of dump. Cards in the systems like L-, N-, V-Class and newer all support this. Details can be found in KMINE document S3100004913.

A swap device can also be used as dump device in order to save disk space but there are two disadvantages:

- 1) Is the **primary** swap device (usually `/dev/vg00/lvol2`) also configured as dump device, it takes more time for the system to bootup after a systemcrash.
Reason: When a dump is found on the dump device during startup it will be written to the local filesystem (by the `rc` command `savecrash`). In the case that the dump device is also the primary swap, `savecrash` cannot run in the background because the swap area may be used during further startup.
- 2) Were there any problems with `savecrash` (lack of space in the crash directory) you still

have the possibility to run it again after the system boot completed (-r Option for resave dump). In case of a swap device there is a risk that parts of the dump are overwritten by "swapping" activities and therefore unusable.

You can influence the interaction of savecrash/core and swapon in the config file of savecrash/core. (see manpage of savecrash/core -w option)

Configuration steps

Creating the logical volumes that should be used for dump

You can specify up to 32 different dump devices. Each dump logical volume has to be **contiguous**, i.e. all physical extents are placed one after another and reside on a single PV. Such a LV can be created with the option -C y of lvcreate command. **Bad block relocation** must be disabled (-r n):

```
# lvcreate -L <size in MB> -n lvdump -C y -r n /dev/vg00
```

You can check the LV parameters with lvdisplay:

```
# lvdisplay /dev/vg00/lvdump | grep Allocation
Allocation                strict/contiguous
```

```
# lvdisplay /dev/vg00/lvdump | grep Bad
Bad block                 off
```

The dump LVs must not contain a filesystem of course.

Activating these logical volumes, i.e. tell the system to use them for dump

A *traditional dump LV* has to be located in the root VG (vg00) and the `lvlnboot` command is used to tell the system to use these LVs for dump. A reboot is necessary in order to activate them. Here's how to configure such a dump device:

Display the current settings:

```
# lvlnboot -v
Boot Definitions for Volume Group /dev/vg00:
Physical Volumes belonging in Root Volume Group:
  /dev/dsk/c0t6d0 (10/0.6.0) -- Boot Disk
  /dev/dsk/c0t5d0 (10/0.5.0)
Root:  lvol1  on:  /dev/dsk/c0t6d0
Swap:  lvol2  on:  /dev/dsk/c0t6d0
No Dump Logical Volume configured
```

Option -d sets the dump device:

```
# lvlnboot -d lvol2 /dev/vg00
# lvlnboot -d lvdump /dev/vg00
```

Check it:

```
# lvlnboot -v | grep dump
Dump:  lvol2  on:  /dev/dsk/c0t6d0, 0
```

```
Dump: lvdump      on:      /dev/dsk/c0t6d0, 1
```

If the dump devices are configured according your needs you have to reboot in order to make the changes take effect. The message buffer displays all valid dumpdevices during reboot:

```
# dmesg | grep DUMP
Logical volume 64, 0x2 configured as DUMP
Logical volume 64, 0x9 configured as DUMP
```

If you like to use a dump device for other purposes you have to deconfigure it using `lvrmboot`. Only the last dump device can be deconfigured:

```
# lvrmboot -d lvdump /dev/vg00
```

NOTE: An entry in the kernel (`/stand/vmunix`) is necessary if you like to have more than one (traditional) dump device with LVM. This entry is set by default:

```
# strings /stand/vmunix | grep "dump lvol"
dump lvol
```

As of UX 11.00 you have the possibility to configure additional dump devices online, i.e. without the need of a reboot. These dump LVs must not be configured using `lvlnboot -d` but with `crashconf(1M)`. You are no longer restricted to choose a dump LV from the root VG only. The configuration of such dump devices is similar to the configuration of secondary swap devices. Here's how to configure a dump device online:

Add a line for each dump device to `/etc/fstab`, e.g.:

```
/dev/vg01/lvdump / dump defaults 0 0
```

Then run `crashconf -a` to activate it and `crashconf` to verify that it is enabled. Configuring non-root dump devices is similar to configuring secondary swap devices.

Refer to the `crashconf(1m)` and `fstab` manual pages for details.

NOTE: Whenever you have dump devices that are not also used for swap activity, make sure that they are configured last. This will cause them to be used first (dump goes from the end backward), which will minimize the chance of writing into an area shared by swap. Writing into swap space is undesirable because it will slow down your reboot processing (see section above).

NOTE: There are often questions like: "Why is the dump LV not mirrored like root, boot and swap LVs are?"

```
# lvlnboot -v
Boot Definitions for Volume Group /dev/vg00:
Physical Volumes belonging in Root Volume Group:
    /dev/dsk/c0t6d0 (10/0.6.0) -- Boot Disk
    /dev/dsk/c0t5d0 (10/0.5.0) -- Boot Disk
Root: lvol1      on:      /dev/dsk/c0t6d0
      lvol2      on:      /dev/dsk/c0t5d0
Swap: lvol2      on:      /dev/dsk/c0t6d0
```

```

                                /dev/dsk/c0t5d0
Dump: lvol2      on:          /dev/dsk/c0t6d0, 0
Dump: lvdump    on:          /dev/dsk/c0t6d0, 1

```

The answer: the system dumps onto a previously configured area of the disk. The dump process is a low level routine that bypasses the LVM layer, hence the data is not going to be mirrored. The OS simply stored the hardware path of the disk and the starting and ending offset on this disk at the time you activated it. This information is given by the dump LV. This is the reason why dump LVs must be contiguous.

The dump/savecrash process

Writing the memory image to the dump devices

The kernel routine responsible for dumping is `dumpsys()`.

Dump formats

There are four known dump formats. Which format you deal with can be found in the INDEX file (`grep version INDEX`):

COREFILE (Version 0)

This format, used up through HP-UX 10.01, consists of a single file containing the physical memory image, with a 1-to-1 correspondence between file offset and memory address. Normally there is an associated file containing the kernel image. Sources or destinations of this type must be specified as two pathnames to plain files, separated by whitespace; the first is the core image file and the second is the kernel image file.

COREDIR (Version 1)

This format, used in HP-UX 10.10, 10.20, and 10.30, consists of a `core.n` directory containing an INDEX file, the kernel (`vmunix`) file, and numerous `core.n.m` files, which contain portions of the physical memory image.

CRASHDIR (Version 2)

This format, used in HP-UX 11.00 and later, consists of a `crash.n` directory containing an INDEX file, the kernel and all dynamically loaded kernel module files, and numerous `image.X.Y` files, each of which contain portions of the physical memory image and metadata describing which memory pages were dumped and which were not.

PARDIR (Version 5)

This format is used in UX 11.11 and later. It is very similar in structure to the CRASHDIR format in that it consists of a `crash.n` directory containing an INDEX file, the kernel and all dynamically loaded kernel module files, and numerous `image.X.Y` files, each of which contain portions of the physical memory image and metadata describing which memory pages were dumped and which were not. In addition to the primary INDEX file, there are auxiliary index files (`indexX.Y`), that contain metadata describing the image files containing the memory pages. This format will be used when the [dump is compressed](#). See `crashconf(1M)`.

Other formats, for example tape archival formats, may be added in the future.

Selective dumps

The most significant change compared to UX 10.X is the possibility of configuring **selective dumps**. Dumps no longer contain the entire contents of physical memory. With memory sizes growing in leaps and bounds, it become critical that HP-UX dump only those parts of physical memory which are considered useful in debugging a problem. By default you get a core of approx. 5-40% of physical memory, varying with the state of the system at dump time. Configuration can be checked and modified with the crashconf utility:

```
# crashconf
```

CLASS	PAGES	INCLUDED IN DUMP	DESCRIPTION
UNUSED	14253	no, by default	unused pages
USERPG	23876	no, by default	user process pages
BCACHE	129981	no, by default	buffer cache pages
KCODE	2044	no, by default	kernel code pages
USTACK	451	yes, by default	user process stacks
FSDATA	753	yes, by default	file system metadata
KDDATA	72447	yes, by default	kernel dynamic data
KSDATA	17699	yes, by default	kernel static data

```
Total pages on system:          261504
```

```
Total pages included in dump:    91350
```

DEVICE	OFFSET(kB)	SIZE (kB)	LOGICAL VOL.	NAME
31:0x006000	72544	524288	64:0x000002	/dev/vg00/lvol2
		524288		

Compressed dumps

Even with selective dump feature a Superdome equipped with 256GB RAM would take hours to write the dump to the dump devices. The bottleneck of copying system memory to disk is the I/O path. This could be alleviated by dumping to multiple disks in parallel but the system firmware (IODC) isn't designed to permit multiple simultaneous I/O requests. Thus the only approach is to limit the amount of I/O that has to be done.

There is a new feature called *compressed dumps* available as of HP-UX Itanium release UX 11i v2 (i.e. UX 11.23) and additionally for UX 11i v1 (i.e. UX 11.11). The data is compressed (using LZO algorithm) before being written out to the dump device. When the system crashes, the dump subsystem assigns one processor to perform the writes to the dump device(s). It assigns another four processors to perform compression.

The dump compression features is targeted for large memory systems. Following requirements must be met:

Systems: Superdome, Keystone, Matterhorn and Prelude

OS: PA-RISC: UX 11i v1 (11.11) + patch
Itanium: UX 11i v2 (11.23)

Configuration: at least 2GB RAM,
at least 5 processors

The compression option is turned ON by default. But it just a hint to the kernel. At the time of a system crash, the dump subsystem examines the state of the system and its resources to determine whether it is possible to use compression. Depending on the resources available, the system decides dynamically whether to dump compressed or not.

Other situations can cause the dump subsystem to decide not to dump compressed: recursive panic, memory allocation failure - all logged on system console at crash dump and flagged in the kernel.

HP can't guarantee a specific compression factor. All compression tends to be dependent on the type of data being compressed, in particular how random it is. The dump should speedup by at least a factor of 3 with default selective dump configuration. More typically, customers will experience a factor of 7.

The crashconf(1M) command was enhanced to be able to configure dump compression:

```
# crashconf -c on

# crashconf -v
CLASS          PAGES    INCLUDED IN DUMP  DESCRIPTION
-----
UNUSED         3645411  no, by default    unused pages
USERPG         7113     no, by default    user process pages
BCACHE         210990   no, by default    buffer cache pages
KCODE          2670     no, by default    kernel code pages
USTACK         264      yes, by default   user process stacks
FSDATA         116      yes, by default   file system metadata
KDDATA         68736    yes, by default   kernel dynamic data
KSDATA         259004   yes, by default   kernel static data

Total pages on system: 4194304
Total pages included in dump: 328120

Dump compressed: ON
```

DEVICE	OFFSET(kB)	SIZE (kB)	LOGICAL VOL.	NAME
31:0x03a000	310112	4194304	64:0x000002	/dev/vg00/lvol2
		4194304		

If you like to make the configuration changes either for selective dump or for compressed dumps resistant across reboots you need to modify the rc-script `/etc/rc.config.d/crashconf`. Usually there should be no need to change the defaults.

The compressed dump feature uses a new crash dump format, [PARDIR](#), for saving the dumps. You recognise a compressed dump with this evidences:

- In the INDEX file you will find a version 5.
- In the dump directory you will find `indexX.Y` files along with the usual `image.X.Y` files.

The dumpreading tools (p4, crashinfo, kmeminfo, etc...) are aware of this new format.

Since the dump is compressed you have little gain to compress it again with gzip, yet since

the compression is done with a 'compress(1)' compatible algorithm and small chunks, gzip'ing the dump still reduce it a bit sometime.

A consequence of the compressed dump is indeed a faster "time to dump" and a somewhat faster "time to reboot" but the dumpreading tools suffer a serious performance penalty, making the "time to diagnose" or "time to fix" significantly longer.

NOTE: To enable compressed dump feature at UX 11.11 you need to install the CDUMP11i product from http://www.software.hp.com/ER_products_list.html. This product contains a set of enabling patches. At UX 11.23 the compressed dump feature is enabled in core, hence no product or patches are needed.

Documentation about the compressed dump feature can be found at:
<http://www.docs.hp.com/hpux/os/11i/index.html#System%20Administration>

Saving the dump to the filesystem

After the system has finished to write the whole or only parts of the dump to the dump devices, the system reboots and automatically starts up again. When booting up, the system starts a rc script to copy the dump into the file system.

As of UX 11.00 the rc script itself is /sbin/init.d/savecrash. The configuration file is stored at /etc/rc.config.d/savecrash. The default location is /var/adm/crash with sub directories named crash.n for every saved crash. The crash.n directory contains an ASCII file named INDEX that contains some metadata of the dump, a copy of the current kernel vmunix and files for every saved contiguous chunk of memory named image.m.n. If the kernel contains loadable modules, those are copied to the dump directory too.

You can configure crash directory, compression mode, etc. in the appropriate configuration file /etc/rc.config.d/savecrash:

Here are the most important options:

SAVECRASH	1 = save a crashdump (default) 0 = do not save a crashdump
SAVECRASH_DIR	directory for the crashfiles. Default is /var/adm/crash
COMPRESS	0 = never compress 1 = always compress 2 = compress in case of insufficient space in crasdirectory (default)

Further options (MINFREE, SWAP_LEVEL, CHUNK_SIZE, SAVE_PART, FOREGRD, LOG_ONLY) are explained in the comments of the config file.

Saving the dump manually

If the dump was not saved completely due to lack of space in the crash directory you have the possibility to save the dump again. The `-r` option (resave) need to be included when this is not the first time that savecrash runs.

```
# savecrash -v [-r] <crash directory>
```

There is also the possibility to save the dump directly to a DDS tape:

```
# savecrash -v [-r] -t /dev/rmt/0m
```

Analysis of the dump

A complete analysis of a crashdump requires deep internal knowledge and much experience. That would certainly go beyond this document. Here I'd like to explain how to use the utility *crashinfo* in order to narrow down the cause of the crash.

If you like to examine the dump by yourself, please refer to the excellent online webcourse offered by the Expert Center. This course should be considered as starting point for any dump analysis. Whenever you deal with a crashdump i recommend you to visit this site. In most cases you should be able to find a solution. Links to all available dump reading tools are included.

<http://wtec.cup.hp.com/~hpux/crash/FirstPassWeb/> (HP internal)

About the crashinfo utility

crashinfo is an executable that is based on libp4, the library of the [P4 kernel debugger](#). It replaces the old whathappened perl script that was based on the [Q4 kernel debugger](#). P4 and crashinfo are much more powerful and advanced than Q4/whathappened. The P4 debugger is based on the korn shell (ksh88) which makes it comfortable to use and the libp4 library.

p4 and crashinfo can be performed on a dump (by executing it from within a crash directory) as well as on a live system (by executing it not within a crash directoy). The latter can be useful to examine kernel structures when the system is e.g. not completely hung.

crashinfo is smart, depending on the type of the crash (PANIC, TOC, SG TOC or HPMC) it prints out the appropriate structures. It also reacts to certain conditions e.g. system low on free memory, spinlock panics, etc. and prints out the necessary data.

How to obtain and install and execute crashinfo

Use the standalone version to perform a quick check

Obtain the *static (standalone)* version of the crashinfo binary from the Ktools server

<http://ktools.france.hp.com/~ktools/cgi-bin/p4ooshop-1t.cgi?tl=crashinfo> (HP internal)

From there the tool can be sent to the customer by email or pushed to an external ftp server. Size is about 800K.

Store the static crashinfo binary e.g. at `/usr/contrib/bin/` on the affected system.

This version should be used to perform a first quick check of the dump.

To get a fingerprint of the dump simply run the standalone crashinfo without options from within the crash directory:

```
# cd /var/adm/crash/crash.0
# /usr/contrib/bin/crashinfo >ci.out
```

NOTE: Should the chunkfiles (image.n.m for a UX 11.X crashdump) be compressed (the suffix .gz indicates that) they get decompressed automatically during the execution of crashinfo. This can take a while. Be sure to have enough space left in the crash directory.

With the help of the webcourse mentioned above it should be possible to solve most of the problems.

Anyway in some cases you might need information that is beyond the standard output of crashinfo. In this case you can use one of crashinfo's options or use the *P4 debugging environment* to perform a deeper analysis.

At this point you have to decide whether to ship the debugging tools to the customer and provide a remote connection for a HP RCE or to ship the dump to the Response Center, either via ftp upload or on a DDS tape or CD-ROM by mail. Which of the above possibilities (remote login, ftp upload, ship by mail) is appropriate depends on availability of remote login, the size of the dump and the severity of the problem.

If remote login is not possible, ship the dump to the Response Center

If you choose to analyze the dump on the customers system, obtain the tools either from the Ktools server:

<http://ktools.france.hp.com/~ktools/cgi-bin/p4ooshop.cgi> (HP internal)

(select *p4*, then *shared*, then *internet*)

or via anonymous ftp from:

ftp://tahoe.grc.hp.com/dumpreading/dump_analyse.tar.gz (HP internal)

Size is about 7MB.

Unpack the files e.g. below /usr/contrib/dumpreading on the system, where the dump is located.

Before starting you need to set some path variables:

```
export P4_ROOT=/usr/contrib/dumpreading
export PATH=$P4_ROOT/bin:$P4_ROOT/p4:$PATH
export SHLIB_PATH=$P4_ROOT/bin:$P4_ROOT/p4
```

Either put the above lines in /etc/profile or simply source the included set_env file in order to set these variables:

```
# cd /usr/contrib/dumpreading
# . ./set_env
```

If remote login is not possible, ship the dump to the Response Center

If you choose to ship the dump to the Response Center additional information from the customers system depending on the type of the crash is needed.

Examine crashinfo output to determine which of the [crash event types](#) this is:

PANIC	the system ran into an unhandable condition and panicked
TOC	the system was hung and you TOCed it
SG TOC	the TOC was initiated by MC/ServiceGuard
HPMC	High Priority Machine Check. The crash was caused by a HW failure

Example:

```
# cat ci.out | grep "Note: Crash"
Note: Crash event 0 was a PANIC !
```

Provide the following:

```
swlist -l product >swlist.out
```

(currently installed software & patches)

/var/adm/syslog/OLDSyslog.log (the syslog from the previous boot)

Additionally in case of a *TOC*, i.e system hang answer these questions:

- ”Did you try a telnet connection to the system? How exactly did it fail?”
- ”Did you try a rlogin connection to the system? How exactly did it fail?”
- ”Did you try a console connection to the system? How exactly did it fail?”
- ”Did the system respond to ping?”
- ”What was the value shown on the hex display?”

Additionally in case of a *ServiceGuard TOC*:

/var/adm/syslog/[OLD]syslog.log (appropriate syslogs of **all** nodes in the cluster)

Additionally in case of a *HPMC*:

/var/tombstones/ts99 (tombstone file containing chassis logs and PIM data)
system’s serial number (obtained from MP/GSP)

Answering the following questions is very important, too:

- ”Did the system hang or panic more than once recently?”
- ”Did anything change recently?” (e.g. kernel patches installed, 3rd party software installed, configuration changes or simply a reboot.

NOTE: A system that panics/hangs multiple times although no changes have been performed is likely to suffer from a hardware problem. Whereas hardware failures can happen all of a sudden, software failures are usually caused by configuration changes.

Please log a **hardware case** when your system crashed due to HPMC,
else log a **software case**.

Analysis beyond standard crashinfo output

crashinfo’s options

crashinfo has some options that might be useful:

```
$ crashinfo -h
crashinfo (3.19)
Usage: crashinfo [options ...] [coredir | kernel core]
Default: coredir="." if "INDEX" file present else
        kernel="/stand/vmunix" core="/dev/kmem"
Options:
  -h | -help [flag,flag...]
        flags: detail
  -u | -update
  -v | -verbose
  -c | -continue
  -H | -Html
  -e | -email <mail_addr>[,flag,flag...]
        flags: file=<file>
              from=<from>
              callid=<callid>
  -t | -trace [flag,flag...]
        flags: args
              regs      (PA Only)
```

```

Regs      (PA Only)
locals    (IA64 Only)
frame     (IA64 Only)
mems      (IA64 Only)
bsp       (IA64 Only)
ss        (IA64 Only)

-s | -syscall
-f | -full_comm
-l | -listonly
-n | -nolist
-S | -Sleep
-i | -ioscan
    -ofiles [pid]
    -signals [pid]
    -vmtrace [flag,flag...]
        flags: bucket=<bucket>
              arena=<arena>
              count=<num>
              leak
              cor
              log
              parse
-kmeminfo

```

Refer to the [crashinfo homepage](http://crashinfo.hp.com) in order to get more information on the usage.

Working with the P4 debugger

From within the dump directory execute p4:

```

$ p4
Send bugs, remarks, ideas to --> ktools@wtec.cup.hp.com

Web based p4 at http://ktools.france.hp.com/~ktools/wp4

$ man          # For online help on p4 functions
$ man -l       # For a listing of p4 functions
$ ref -n       # Lookup p4 reference manual on the web -
               http://ktools.france.hp.com/~ktools/p4-4/
$ p4 -u        # Get the latest version of p4

P4 revision: 7.103

Loading symbols from lab07/vmunix
Kernel TEXT pages not requested in crashconf
Will use an artificial mapping from lab07/vmunix TEXT pages

Using a.out from lab07/vmunix and mem from lab07/INDEX ...
Open lab07/vmunix and lab07/INDEX OK

HP-UX trefftz1 B.11.00 U 9000/800 648359312
This is a WIDE mode kernel (LP64)

/dumps/dumpread/group1$

```

To obtain a stacktrace:

```

$ trc event 0
Event #0 : proc[29] pid=1498 tid=1555 cmd="/usr/sbin/nfsd 4"
===== EVENT =====
= Event #0 is PANIC on CPU #3
= p crash_event_t 0x22000
= p rpb_t 0x975608
= Using pc from pim.wide.rp_rp_hi = 0x3a1174

```

```

===== EVENT =====
panic+0x14
report_trap_or_int_and_panic+0x84
trap+0xe14
thandler+0xd24
+----- TRAP -----
| Trap type 6 in KERNEL mode at 0 (0x00000000_00000000)
| p struct save_state 0xa8da000.0x400003ffffff2850
+----- TRAP -----
suspicious trap addr, try to resync with ss_rp=0x277a48
sendfile_rele+0x318
...

```

P4 includes a pool of useful commands:

```

$ man -l
p4_btype_def      - Define a new base type for p4
p4_ls_type        - List all the p4 data types
p4_kernel_symbols - Access kernel global variables as ksh variables
p4_ls_su          - List all struct/union data types
p4_ls_td          - List all typedefs
p4_ls_enum        - List all enum types or members of an enum type
p4_add_enum       - Define an enum type or enumerant
p4_struct_init    - Dynamically load additional debug infos
p4_print          - General print utility
p4_print_next     - General print utility, print next element
p4_print_prev     - General print utility, print previous element
p4_print_redo     - Redo p4_print command
p4_printf         - Print formatted output similar to printf(3S)
...

```

Each command has a man page.

Some P4 commands are intended to provide the same functionality as existing HP-UX commands. The usually begin with a capital letter:

```

$ Bdf
Filesystem      kbytes    used    avail %used Mounted on
/dev/root        204800    75635   121232   38% /
/dev/vg00/lvol1  299157    46240   223001   17% /stand
/dev/vg00/lvol8  4706304  1018611 3459291   23% /var
/dev/vg00/lvol7  1343488   590072   706389   46% /usr
/dev/vg00/lvol4   204800   146089   55072    73% /tmp
/dev/vg00/lvol6  1024000   966638   53817   95% /opt
/dev/vgdata/lvdata 102400000 78978488 23238632   77% /mnta2
/dev/vgabin/lvbin 102400000 29787064 72059592   29% /mnta1
/dev/vgprog/vgprog 102400000 100095664 2227558   98% /interconnect
/dev/vg00/lvol5    20480    14763   5396    73% /home
/dev/vg03/lvdata2 102400000 38024176 63930960   37% /mnta3
/opt/bmpa/tmp/.MTP_interface_pipe.15840
                                0         0         0         0%

```

```

$ Swapinfo -tm
      Mb      Mb      Mb      PCT      START/      Mb
TYPE    AVAIL    USED    FREE    USED    LIMIT RESERVE  PRI  NAME
dev      4096     498     3598    12%      0      -      1  /dev/vg00/lvol2
reserve  -      1003    -1003
memory   1580     598     982     38%
total    5676    2099    3577    37%      -      0      -

```

```

hprtdu22[dumpy]:/dumps/dumpread/labs/lab20$ BootString
disc(10/4/12.0.0;0)/stand/vmunix

```

```

hprtdu22[dumpy]:/dumps/dumpread/labs/lab20$
hprtdu22[dumpy]:/dumps/dumpread/labs/lab20$ Boottime
0x3d27fc5f : Sun Jul 7 10:31:27 2002
hprtdu22[dumpy]:/dumps/dumpread/labs/lab20$
hprtdu22[dumpy]:/dumps/dumpread/labs/lab20$ Time
0x3d2d41de : Thu Jul 11 10:29:18 2002
hprtdu22[dumpy]:/dumps/dumpread/labs/lab20$
hprtdu22[dumpy]:/dumps/dumpread/labs/lab20$ Crashconv -v
p4: Crashconv: not found
hprtdu22[dumpy]:/dumps/dumpread/labs/lab20$ Crashconf -v

```

CLASS	PAGES	INCLUDED	IN DUMP	DESCRIPTION
UNUSED	24611	no,	by default	unused pages
USERPG	95002	no,	by default	user process pages
BCACHE	162582	no,	by default	buffer cache pages
KCODE	1908	no,	by default	kernel code pages
USTACK	1440	yes,	by default	user process stacks
FSDATA	1258	yes,	by default	file system metadata
KDDATA	25286	yes,	by default	kernel dynamic data
KSDATA	15593	yes,	by default	kernel static data

```

Total pages on system:          327680 ( 1310720 Kb )
Total pages included in dump:   43577 ( 174308 Kb )

```

DEVICE	OFFSET(Kb)	SIZE (Kb)	LOGICAL VOL.	NAME
28:0x030000	101216	1024000	64:0x000002	/dev/vg00/lvol2
		1024000		

```

Total avail dump space:          1024000 ( 256000 pages )
Space for dump headers:          -      60 (      15 pages )
=====
Total useable dump area:          1023940 ( 255985 pages )

```

```

$ CpuUsage

```

pid	tid	pri	spu	kt_cpu	recent	user	sys	intr	kt_start
p_comm									
0	0	128	3	0	0	0	4695	162	0x3d27fc5f
swapper									
1	1	168	0	0	0	265	5052	0	0x3d27fc68 init
2	2	128	0	0	0	0	1104	0	0x3d27fc5f
vhand									
3	3	128	2	1	0	0	93680	1865	0x3d27fc5f
statdaemon									
...									

```

$ Dmesg
o
10/0 c720
10/0.6 tgt
10/0.6.0 sdisk
10/0.7 tgt
10/0.7.0 sctl
...

```

```

$ Fstyp /var
hfs
f_bsize: 8192          /* preferred file system block size */
f_frsize: 1024         /* fundamental file system block size */
f_blocks: 1443040      /* total blocks of fr_size on file system */
f_bfree: 532045        /* total number of free block in fs */
...

```

```

$ Ipcstatus -m
IPC status from /dumps/dumpread/labs/lab20 as of Thu Jul 11 10:29:18 2002

```

```

T      ID      KEY      MODE      OWNER      GROUP
Shared Memory:
m      0 0x411057d6 --rw-rw-rw-    root      root
m      1 0x4e100002 --rw-rw-rw-    root      root
m      2 0x41142787 --rw-rw-rw-    root      root
m      3 0x5011e167 --r--r--r--    root      other
m     9220 0x0c6629c9 --rw-r-----    root      root
...

```

```

$ Processes
Loaded 4116 proc_t entries in 'DefaultView'

```

```

$ keep p_stat (UX 10.X and 11.00 only)
Kept 281 entries in DefaultView

```

```

$ vp p_pid p_ppid p_comm | grep getty
0x00000663 0x00000001 getty

```

```

$ Ps -p 16440
Sleep PRI TID PID PPID PCOMM SC_NAME KSTAT CTXT_FLAGS
1026 661 12314 16440 23369 rm unlink TSSLEEP 0x00000000

```

Additionally there are other useful commands:

Get the command line of a process:

```

$ pcmd -p 16440
addr pindx pid : command
0x6393d80 2225 16440 : rm 1_450.dbf 1_4500.dbf 1_45000.dbf 1_45001.dbf
1_45002.dbf

```

Get the stacktrace of a process:

```

$ trace -a -p 16440
proc[2225] pid=16440 tid=12314 cmd="rm 1_450.dbf 1_4500.dbf 1_45000.dbf 1_45"
Process : p proc_t 0x6393d80
proc[2225] pid=16440 rm
Kthread : p kthread_t 0x67e89a8
Using PCB: p user_t 0x627f400.0x400003ffffff0000
SR5=0x0627f400
SP SZ RP Return Name
0x400003ffffff21a0 0x00c0 0x00128a8c _swtch+0xd4
arg0: 0x000000000001cc988
0x400003ffffff20e0 0x0130 0x001286ac _sleep+0x154
arg0: 0x000000000000957448
arg1: 0x00000000000000295
0x400003ffffff1fb0 0x00d0 0x001cc988 getnewbuf_desperate+0x258
arg0: 0x00000000000000001
arg1: 0x000000000000002000
0x400003ffffff1ee0 0x0120 0x00168d2c getnewbuf+0x584
arg0: 0x000000000000004850
arg1: 0x000000000000002000
...

```

Print values and structures:

Print value at address 0x023ff070:

```

$ p i4 0x023ff070
0x023ff070
0x023ff070 : 0x023e95f0

```

I.e. the value referenced by the "pointer" 0x023ff070 is 0x023e95f0

If you know that you are referencing a certain structure you can print it:

```
$ p struct inode 0x023ff070
0x023ff070
0x023ff070 :
0x023ff070 struct inode {
0x023ff070     struct inode *i_chain[2];           0x023e95f0
0x023ff078     dev_t     i_dev;                   0x40000005
0x023ff07c     ino_t     i_number;                 0x00058a40
0x023ff080     u_int     i_flag;                   0x00000446
0x023ff084     ushort    i_lockword;              0x0011
0x023ff088     tid_t     i_tid;                   0x00001b2f
0x023ff08c     struct vnode {
0x023ff08c         u_short v_flag;                0x0000
...
```

P4 provides some nice commands to **calculate**:

convert to decimal:

```
$ d 0x100
256

or

$ dec 0x100
256
```

convert to hexadecimal:

```
$ x 256
0x100

or

$ hex 256
0x100
```

convert to any format:

```
$ Let -b 256
100000000

$ x 0x7fff64d8-0x30
```

Print kernel globals/tunables:

```
$ printf '%d\n' nproc
6420

$ d nproc
6420

$ d vxfs_ninode
128000
```

NOTE: dec, hex and Let are aliases for the p4_let(1) command.

crashinfo output example

crashinfo (3.10) output

```
=====
= Table Of Contents =
=====
```

```
* General Information
* Crash Events
* Message Buffer
* Memory Globals
* Buffer Cache Globals
* Swap Information
* Global Error Counters / kmem_writes
* Network Interfaces
* IOVA Usage Check
* Crash Event / Processor Information
* Processor Clock Info
* Syswait Array
* Load Averages
* Thread Information
* Kernel Patches
```

```
=====
= General Information =
=====
```

Dump time Fri May 9 08:14:12 2003 UTC-2
System has been up 1 minute.

```
System Name      : HP-UX
Node Name       : banana
Model          : 9000/800/A500-7X
HP-UX version   : B.11.00 (64-bit Kernel)
Number of CPU's : 2
Disabled CPU's  : 0
CPU type       : PCXW+ (750 Mhz)
CPU Architecture : PA-RISC 2.0
Load average    : 0.29 0.08 0.03
```

```
=====
= Crash Events =
=====
```

Note: Crash event 0 was a TOC !

Note: This seems to be a user initiated TOC !
It seems the monarch processor has not updated the system wide clock
for approx 4547 seconds. Concentrate on the stack trace for the monarch
processor (usually CPU 0) !
For more information go to:
http://wtec.cup.hp.com/~hpux/crash/FirstPassWeb/PA/toc/crashinfo_clock.htm

Stack Trace for crash event 0
=====

```
===== EVENT =====
= Event #0 is TOC on CPU #0
= p crash_event_t 0x22000
= p rpb_t 0x7bc358
= Using pc from pim.wide.rp_pcoq_head_hi = 0x126348
===== EVENT =====
SR4=0x00000000
      SP      RP Return Name
0x00000000b7e22a0 0x00126348 idle+0x1000
0x00000000b7e2050 0x00128adc swidle+0x20
```

Stack Traces for other processors
=====


```

Processor #1
===== EVENT =====
= Event #1 is TOC on CPU #1
= p crash_event_t 0x22030
= p rpb_t 0xcac370
= Using pc from pim.wide.rp_pcoq_head_hi = 0x126388
===== EVENT =====
SR4=0x00000000
      SP      RP Return Name
0x000000000b7e52a0 0x00126388 idle+0x1040
0x000000000b7e5050 0x00128adc swidle+0x20

=====
= Message Buffer =
=====

gate64: sysvec_vaddr = 0xc0002000 for 1 pages
NOTICE: autofs_link(): File system was registered at index 3.
NOTICE: nfs3_link(): File system was registered at index 5.
0 sba
0/0 lba
0/0/0/0 btlan3
0/0/1/0 c720
0/0/1/0.7 tgt
0/0/1/0.7.0 sctl
0/0/1/1 c720
0/0/1/1.7 tgt
0/0/1/1.7.0 sctl
0/0/1/1.15 tgt
0/0/1/1.15.0 sdisk
0/0/2/0 c720
0/0/2/0.7 tgt
0/0/2/0.7.0 sctl
0/0/2/1 c720
0/0/2/1.7 tgt
0/0/2/1.7.0 sctl
0/0/2/1.15 tgt
0/0/2/1.15.0 sdisk
0/0/4/1 asio0
0/2 lba
0/4 lba
0/6 lba
8 memory
160 processor
162 processor
btlan3: Initializing 10/100BASE-TX card at 0/0/0/0....

System Console is on the Built-In Serial Interface
Entering cifs_init...
Initialization finished successfully... slot is 8
Logical volume 64, 0x3 configured as ROOT
Logical volume 64, 0x2 configured as SWAP
Logical volume 64, 0x2 configured as DUMP
Swap device table: (start & size given in 512-byte blocks)
entry 0 - major is 64, minor is 0x2; start = 0, size = 8388608
Dump device table: (start & size given in 1-Kbyte blocks)
entry 0 - major is 31, minor is 0x1f000; start = 310112, size = 4194304
Warning: file system time later than time-of-day register

Getting time from file system
Starting the STREAMS daemons-phase 1
Create STCP device files
Starting the STREAMS daemons-phase 2
B2352B/9245XB HP-UX (B.11.00) #1: Wed Nov 5 22:38:19 PST 1997

Memory Information:
physical page size = 4096 bytes, logical page size = 4096 bytes
Physical: 3145728 Kbytes, lockable: 2374088 Kbytes, available: 2731304 Kbytes

=====
= Memory Globals =
=====

```

```
Physical Memory      = 786432 pages (3.00 GB)
Free Memory          = 676440 pages (2.58 GB)
Average Free Memory  = 599788 pages (2.29 GB)
desfree              = 3072 pages (12.00 MB)
minfree              = 1280 pages (5.00 MB)
```

```
=====
= Buffer Cache Globals =
=====
```

```
dbc_max_pct          = 50 %
dbc_min_pct          = 5 %
dbc_current_pct       = 5.4 %
bufpages              = 42627 pages (166.51 MB)
Number of buf headers = 22596
```

```
fixed_size_cache     = 0
dbc_parolemem         = 0
dbc_stealavg          = 0
dbc_ceiling           = 393216 pages (1.50 GB)
dbc_nbuf              = 19660
dbc_bufpages          = 39321 pages (153.60 MB)
dbc_vhandcredit       = 0
orignbuf              = 0
origbufpages          = 0 pages
```

```
=====
= Swap Information =
=====
```

```
swapinfo -mt emulation
=====
```

	Mb	Mb	Mb	PCT	START/	Mb			
TYPE	AVAIL	USED	FREE	USED	LIMIT	RESERVE	PRI	NAME	
dev	4096	0	4096	0%	0	-	1	LVM vg00/lv2	
reserve	-	3	-3						
memory	2324	24	2300	1%					
total	6420	27	6393	0%	-	0	-		

```
=====
= Global Error Counters / kmem_writes =
=====
```

```
default_disk_ir = 1
```

Note: Immediate reporting for SCSI devices switched on per default !

```
=====
= Network Interfaces =
=====
```

Name	PPA	Driver	Interface	Mac	States	IP
		Name	Description	Address	Link IP	Address
lan0	0	btlan3	100BT PCI Built-in	0x00306e26clac	UP	n/c n/c

n/c : means "Not Configured", ifconfig has not been done on this interface

If you want more information, you can use : "lanshow -f"

```
=====
= IOVA Usage Check =
=====
```

99% of IOVA still available/free.

```
=====
= Crash Event / Processor Information =
=====
```

Number of processors = 2

s

```

      t
      a
evt  cpu  t  type  dpth  src  cr15      eirr      ipsw
-----
0    0    E  TOC   2      rpb  c600000000000000  0800000000000012  080efc1f  WBCVRQPD1
      mpi  ffffffff0fffffff
1    1    E  TOC   0      rpb  ffffffff0fffffff  0000000100000000  0804fc1f  WCRQPD1

```

Outstanding external interrupts
=====

```

      eirr
cpu  bit   SPL      Handler SPL      Handler
-----
0    4     SPL6/SPINLOCK_EIEM  SPL6/SPINLOCK_EIEM  clock_int
0    59    SPL6/SPINLOCK_EIEM  SPL5/SPLIO          sapic_interrupt
0    62    SPL6/SPINLOCK_EIEM  SPL5/SPLIO          sapic_interrupt
1    31    SPLNOPREEMPT        SPLNOPREEMPT        take_a_trap

```

SPL/EIEM values:

```

0xffffffffffffffff = SPLPREEMPTOK - Default user mode SPL level.
0xffffffff0fffffff = SPLNOPREEMPT - Disable kernel preemption (scheduling interrupt off).
0xffffffff00fffffff = SPL2 - Disable software interrupt (software triggers off).
0xef00080000000000 = SPL5 - Disable IO modules.
0xc700000000000000 = SPL6+LOCK_RESYNC - Disable hardclock+enable clock-resync.
0xc600000000000000 = SPL6 - Disable hardclock.
0x0000000700000000 = SPL7/PSW_I=0 - Disable the world.

```

```

=====
= Processor Clock Info =
=====

```

```

hardclock_late = 796
itick_per_tick = 7500000
lbolt          = 10644 (0x2994)

```

```

      mpi      interval      clk eiem eirr PSW
cpu  timeinval  timer      delta (ticks) od  0,4  0,4  I
-----
0    0x2e22064a2f  0x3492b9cb4da  -455287      796 1 0  0 1  1
1    0x3494f94b457  0x3494f36f29a    0          0  1 1  0 0  1

```

WARNING: Processor 0 appears to have had clock interrupts held off for approx 4547 seconds. Current SPL = 0xc600000000000000 (SPL6).

```

=====
= Syswait Array =
=====

```

```

cpu iowait
---
1  1

```

Note: This shows the number of threads waiting on buffer I/O. First figure out how long the I/O is outstanding. A good way to do so is by searching in the threads list for processes that have a waitchannel like biowait, ogetblk or swbuf. As a rule of thumb, only consider I/O's outstanding longer than 30 seconds (your mileage may vary).

For more information go to:

"http://wtec.cup.hp.com/~hpux/crash/FirstPassWeb/PA/toc/buffer_hang.htm"

```

=====
= Load Averages =
=====

```

```

avenrun
=====
0.29  0.08  0.03

```

```

real_run
=====
0.144118  0.044052  0.015964

```

```
pwrn ("fast" io wait)
=====
0.429802  0.121551  0.043083
```

```
mp_avenrun
=====
cpu0 : 0.461995  0.135315  0.048295
cpu1 : 0.111926  0.030288  0.010752
```

```
=====
= Thread Information =
=====
```

```
9 Threads ran in the last second
46 Threads ran in the last 5 seconds
47 Threads ran in the last 10 seconds
52 Threads ran in the last minute
89 Threads ran in the last hour
```

```
statdaemon ran 84 ticks ago
```

```
Most Common Wait Channels
=====
```

Wait Channel	count	ticks since run:	
		longest	shortest
vx_inactive_thread_sv	25	6271	171
vx_inactive_thread_sv+0x8	25	6271	171
lvmkd_q	6	225	225
streams_mp_sync	2	6307	6306
streams_blk_sync	2	6307	6306

```
Most Common Sleep Callers
=====
```

Sleep Caller	count	ticks since run:	
		longest	shortest
vx_inactive_thread()	50	6271	171
lvmkd_daemon()	6	225	225
wait1()	3	239	0
biowait()	2	0	0

```
Idle Globals
=====
```

```
candidate_idle_spu = 0
migration_cycles = 0
```

```
Running Threads (TSRUNPROC) and idle Processors
=====
```

TID	PID	PPID	TICKS	TICKS	PRI	SPU	STATE	S	TICKS	NREADY	FR	LO	AL	COMMAND
			SINCE	SINCE					C SINCE					
			RUN	IDLE					MIGR					
				0		0	IDLE	N	171	0	0	0		
				0		1	IDLE	N	203	0	0	0		

```
Note:
```

```
FR: free to run on any processor (candidate for thread migration).
LO: locked (via processor affinity/mpctl) to this processor).
AL: Alpha semaphores misses (special scheduling when miss a sema).
```

```
Threads waiting on cpu (TSRUN) - sorted by cpu/pri/ticks-since-run
=====
```

```
Note: There is 1 thread in TSZOMB stat !
```

```
All Threads - sorted by ticks-since-run
=====
```

TID	PID	PPID	TICKS	PRI	SPU	STAT	SYSCALL	COMMAND	WCHAN
503	446	445	0	148	1	SLEEP	execve	sh	biowait(0x42f8a6d8)
502	445	440	0	152	0	SLEEP	execve	sh	proc[33]+0x1a8
497	440	430	0	158	0	SLEEP	waitpid	nettl	proc[27]
35	33	0	69	138	1	SLEEP	n/a	vxfsd	vx_ifree_thread_sv
33	33	0	71	138	0	SLEEP	n/a	vxfsd	
vx_event_wait(0x42b88aa0)									
34	33	0	71	138	0	SLEEP	n/a	vxfsd	vx_iflush_thread_sv
36	33	0	71	138	1	SLEEP	n/a	vxfsd	
vx_inactive_cache_thread_sv									
4	4	0	84	128	0	SLEEP	n/a	unhashdaemon	unhash
3	3	0	84	128	1	SLEEP	n/a	statdaemon	ticks_since_boot
38	33	0	152	138	1	SLEEP	n/a	vxfsd	
vx_logflush_thread_sv									
39	33	0	171	138	1	SLEEP	n/a	vxfsd	vx_attr_thread_sv
75	33	0	171	138	1	SLEEP	n/a	vxfsd	
vx_inactive_thread_sv+0x8									
54	33	0	171	138	0	SLEEP	n/a	vxfsd	
vx_inactive_thread_sv									
37	33	0	171	138	0	SLEEP	n/a	vxfsd	
vx_delxwri_thread_sv									
40	33	0	171	138	0	SLEEP	n/a	vxfsd	vx_tuning_thread_sv
499	442	1	202	127	0	SLEEP	read	nktl_daemon	netdiag_ques+0x44
498	441	440	203	178	0	ZOMB	exit	nettl	
1	1	0	203	168	1	SLEEP	sigsuspend	init	*uptr+0
483	426	1	204	154	0	SLEEP	select	syslogd	selwait
23	23	0	225	147	1	SLEEP	n/a	lvmkd	lvmkd_q
19	19	0	225	147	0	SLEEP	n/a	lvmkd	lvmkd_q
22	22	0	225	147	0	SLEEP	n/a	lvmkd	lvmkd_q
21	21	0	225	147	0	SLEEP	n/a	lvmkd	lvmkd_q
20	20	0	225	147	0	SLEEP	n/a	lvmkd	lvmkd_q
18	18	0	225	147	0	SLEEP	n/a	lvmkd	lvmkd_q
487	430	100	226	158	1	SLEEP	waitpid	nettl	proc[28]
486	429	1	237	155	0	SLEEP	msgrcv	ptydaemon	msgque[0]+0x5c
157	100	1	239	158	0	SLEEP	waitpid	rc	proc[23]
52	33	0	274	138	0	SLEEP	n/a	vxfsd	
vx_inactive_thread_sv									
73	33	0	276	138	1	SLEEP	n/a	vxfsd	
vx_inactive_thread_sv+0x8									
50	33	0	277	138	0	SLEEP	n/a	vxfsd	
vx_inactive_thread_sv									
0	0	0	284	128	0	SLEEP	n/a	swapper	runout
71	33	0	316	138	1	SLEEP	n/a	vxfsd	
vx_inactive_thread_sv+0x8									
69	33	0	319	138	1	SLEEP	n/a	vxfsd	
vx_inactive_thread_sv+0x8									
...									
...									
...									
85	33	0	6271	138	1	SLEEP	n/a	vxfsd	
vx_inactive_thread_sv+0x8									
12	12	0	6306	-32	0	SLEEP	n/a	ttisr	ttirr
28	28	0	6306	100	0	SLEEP	n/a	sblksched	streams_blk_sync
26	26	0	6306	100	0	SLEEP	n/a	smpsched	streams_mp_sync
24	24	0	6306	148	0	SLEEP	n/a	lvmschedd	lv_schedule_daemon
25	25	0	6307	100	1	SLEEP	n/a	smpsched	streams_mp_sync
27	27	0	6307	100	1	SLEEP	n/a	sblksched	streams_blk_sync
10	10	0	6384	100	0	SLEEP	n/a	strweld	weldq_runq
8	8	0	6384	100	0	SLEEP	n/a	supsched	streams_up_runq
9	9	0	6384	100	0	SLEEP	n/a	strmem	__gp+0x4e8
11	11	0	6384	100	0	SLEEP	n/a	strfreebd	str_freeb_idle

```

=====
= Kernel Patches =
=====

```

PHKL_12965	PHKL_13431	PHKL_13810	PHKL_14026	PHKL_14088
PHKL_14763	PHKL_14765	PHKL_15510	PHKL_15547	PHKL_15550
PHKL_15551	PHKL_15553	PHKL_15705	PHKL_15910	PHKL_16074
PHKL_16209	PHKL_16236	PHKL_16819	PHKL_17042	PHKL_17205
PHKL_17258	PHKL_17458	PHKL_17869	PHKL_17953	PHKL_18295
...				
...				
PHNE_15537	PHNE_16017	PHNE_16599	PHNE_17586	PHNE_18272
PHNE_18409	PHNE_19620	PHNE_19759	PHNE_20344	PHNE_20431
PHNE_21217	PHNE_21433	PHNE_21897	PHNE_22086	PHNE_22125

PHNE_22159	PHNE_22244	PHNE_22245	PHNE_22566	PHNE_22642
PHNE_22962	PHNE_23249	PHNE_23456	PHNE_23930	PHNE_24100

Patches related to crash dumps

There are several patches that fix problems related to crash dumps. Either a dump could not be properly or not at all taken or the unwinding of the stack trace was not possible. There have also been problems when saving the crash to the file system or with the crashconf(1M) command. The patches that fix these problems are listed below:

- UX 11.00: [PHKL 20873](#) - 11.00 patch for kernel stack unwinding
[PHKL 21121](#) - 11.00 patch for kernel stack unwinding
[PHKL 21120](#) - 11.00 patch for kernel stack unwinding
[PHKL 20900](#) - 11.00 Add missing crash dump debug information
[PHKL 22926](#) - 11.00 Incomplete Selective Dump, TOC/Panic Failure
[PHKL 20937](#) - 11.00 Fix for TOC vector overwriting
[PHKL 20989](#) - 11.00 Cumulative dump device, dump size patch
[PHKL 20173](#) - 11.00 Include zero page in dumps
[PHKL 20915](#) - 11.00 trap-related panics/hangs
[PHCO 26188](#) - 11.00 savecrash(1M) cumulative patch
[PHCO 20196](#) - 11.00 savecrash startup files cumulative patch
[PHCO 19726](#) - 11.00 crashconf(1M) cumulative patch
- UX 11.11: [PHKL 27918](#) - 11.11 EPIC debug info
[PHKL 25761](#) - 11.11 Fix early dump on Superdome and rp8400
[PHCO 26956](#) - 11.11 savecrash(1M) cumulative patch
- UX 11.22: [PHKL 28787](#) - 11.22 SCSI Partition/Dump driver Cumulative Patch
[PHSS 27990](#) - 11.22 Ultra320 SCSI Patch
[PHKL 28313](#) - 11.22 INIT panic - spinlock held
[PHKL 28720](#) - 11.22 MCA patch
[PHKL 28130](#) - 11.22 IDE driver (ATA/ATAPI/dump) cumulative patch

Additional information

Dump reading webcourse:

<http://wtec.cup.hp.com/~hpux/crash/FirstPassWeb/> (HP internal)

Dump reading webcourse for Itanium systems:

<http://wtec.cup.hp.com/~hpux/crash/ia64crash/> (HP internal)

P4 homepage:

<http://ktools.france.hp.com/~ktools/p4-4/> (HP internal)

There is a nice web based P4:

<http://ktools.france.hp.com/~ktools/wp4> (HP internal)

crashinfo homepage:

<http://www.ukrc.ukr.hp.com/edt/crashinfo.html> (HP internal)

System crash dump white paper:

<http://docs.hp.com/cgi-bin/otsearch/getfile?id=/hpux/onlinedocs/os/syscrash.html>

Refer to the [vPars Chapter](#) to learn how a *virtual partition* system dumps.

Related manual pages:

savecrash(1M), crashconf(1M), crashutil(1M), lvlnboot(1M)