NAME

hwpmc - Hardware Performance Monitoring Counter support

SYNOPSIS

The following option must be present in the kernel configuration file:

options HWPMC_HOOKS

Additionally, for i386 systems:

device apic

To load the driver as a module at boot time:

sysrc kld_list+=hwpmc

Alternatively, to compile the driver into the kernel:

device hwpmc

To enable debugging features (see *DEBUGGING*):

options KTR
options KTR_COMPILE=(KTR_SUBSYS)
options KTR_MASK=(KTR_SUBSYS)
options HWPMC_DEBUG

DESCRIPTION

The **hwpmc** driver virtualizes the hardware performance monitoring facilities in modern CPUs and provides support for using these facilities from user level processes.

The driver supports multi-processor systems.

PMCs are allocated using the PMC_OP_PMCALLOCATE request. A successful PMC_OP_PMCALLOCATE request will return a handle to the requesting process. Subsequent operations on the allocated PMC use this handle to denote the specific PMC. A process that has successfully allocated a PMC is termed an "owner process".

PMCs may be allocated with process or system scope.

Process-scope The PMC is active only when a thread belonging to a process it is attached to is scheduled on a CPU.

System-scope The PMC operates independently of processes and measures hardware events for the system as a whole.

PMCs may be allocated for counting or for sampling:

Counting In counting modes, the PMCs count hardware events. These counts are retrievable using the PMC_OP_PMCREAD system call on all architectures. Some architectures offer faster methods of reading these counts.

Sampling

In sampling modes, the PMCs are configured to sample the CPU instruction pointer (and optionally to capture the call chain leading up to the sampled instruction pointer) after a configurable number of hardware events have been observed. Instruction pointer samples and call chain records are usually directed to a log file for subsequent analysis.

Scope and operational mode are orthogonal; a PMC may thus be configured to operate in one of the following four modes:

Process-scope, counting

These PMCs count hardware events whenever a thread in their attached process is scheduled on a CPU. These PMCs normally count from zero, but the initial count may be set using the PMC_OP_SETCOUNT operation. Applications can read the value of the PMC anytime using the PMC_OP_PMCRW operation.

Process-scope, sampling

These PMCs sample the target processes instruction pointer after they have seen the configured number of hardware events. The PMCs only count events when a thread belonging to their attached process is active. The desired frequency of sampling is set using the PMC_OP_SETCOUNT operation prior to starting the PMC. Log files are configured using the PMC_OP_CONFIGURELOG operation.

System-scope, counting

These PMCs count hardware events seen by them independent of the processes that are executing. The current count on these PMCs can be read using the PMC_OP_PMCRW request. These PMCs normally count from zero, but the initial count may be set using the PMC_OP_SETCOUNT operation.

System-scope, sampling

These PMCs will periodically sample the instruction pointer of the CPU they are allocated on, and will write the sample to a log for further processing. The desired frequency of sampling is set using the PMC_OP_SETCOUNT operation prior to starting the PMC. Log files are configured using the PMC_OP_CONFIGURELOG operation.

System-wide statistical sampling can only be enabled by a process with super-user privileges.

Processes are allowed to allocate as many PMCs as the hardware and current operating conditions permit. Processes may mix allocations of system-wide and process-private PMCs. Multiple processes may be using PMCs simultaneously.

Allocated PMCs are started using the PMC_OP_PMCSTART operation, and stopped using the PMC_OP_PMCSTOP operation. Stopping and starting a PMC is permitted at any time the owner process has a valid handle to the PMC.

Process-private PMCs need to be attached to a target process before they can be used. Attaching a process to a PMC is done using the PMC_OP_PMCATTACH operation. An already attached PMC may be detached from its target process using the converse PMC_OP_PMCDETACH operation. Issuing a PMC_OP_PMCSTART operation on an as yet unattached PMC will cause it to be attached to its owner process. The following rules determine whether a given process may attach a PMC to another target process:

- A non-jailed process with super-user privileges is allowed to attach to any other process in the system.
- Other processes are only allowed to attach to targets that they would be able to attach to for debugging (as determined by p_candebug(9)).

PMCs are released using PMC_OP_PMCRELEASE. After a successful PMC_OP_PMCRELEASE operation the handle to the PMC will become invalid.

Modifier Flags

The PMC_OP_PMCALLOCATE operation supports the following flags that modify the behavior of an allocated PMC:

PMC_F_CALLCHAIN

This modifier informs sampling PMCs to record a callchain when capturing a sample. The maximum depth to which call chains are recorded is specified by the *kern.hwpmc.callchaindepth* kernel tunable.

PMC_F_DESCENDANTS

This modifier is valid only for a PMC being allocated in process-private mode. It signifies that the PMC will track hardware events for its target process and the target's current and future descendants.

PMC_F_LOG_PROCCSW
This modifier is valid only for a PMC being allocated in process-private mode. When this modifier is present, at every context switch, **hwpmc** will log a record containing the number of hardware events seen by the target process when it was scheduled on the CPU.

PMC_F_LOG_PROCEXIT
This modifier is valid only for a PMC being allocated in process-private mode. With this modifier present, **hwpmc** will maintain per-process counts for each target process attached to a PMC. At process exit time, a record containing the target process' PID and the accumulated per-process count for that process will be written to the configured log file.

Modifiers PMC_F_LOG_PROCEXIT and PMC_F_LOG_PROCCSW may be used in combination with modifier PMC_F_DESCENDANTS to track the behavior of complex pipelines of processes. PMCs with modifiers PMC_F_LOG_PROCEXIT and PMC_F_LOG_PROCCSW cannot be started until their owner process has configured a log file.

Signals

The **hwpmc** driver may deliver signals to processes that have allocated PMCs:

SIGIO A PMC_OP_PMCRW operation was attempted on a process-private PMC that does not have attached target processes.

SIGBUS The **hwpmc** driver is being unloaded from the kernel.

PMC ROW DISPOSITIONS

A PMC row is defined as the set of PMC resources at the same hardware address in the CPUs in a system. Since process scope PMCs need to move between CPUs following their target threads, allocation of a process scope PMC reserves all PMCs in a PMC row for use only with process scope PMCs. Accordingly a PMC row will be in one of the following dispositions:

PMC_DISP_FREE	Hardware counters in this row are free and may be use to satisfy either of system scope or process scope allocation requests.
PMC_DISP_THREAD	Hardware counters in this row are in use by process scope PMCs and are only available for process scope allocation requests.
PMC_DISP_STANDALONE	Some hardware counters in this row have been administratively disabled or are in use by system scope PMCs. Non-disabled hardware counters in such a row may be used for satisfying system scope allocation requests.

No process scope PMCs will use hardware counters in this row.

COMPATIBILITY

The API and ABI documented in this manual page may change in the future. This interface is intended to be consumed by the pmc(3) library; other consumers are unsupported. Applications targeting PMCs should use the pmc(3) library API.

PROGRAMMING API

The **hwpmc** driver operates using a system call number that is dynamically allotted to it when it is loaded into the kernel.

The **hwpmc** driver supports the following operations:

PMC_OP_CONFIGURELOG
: Configure a log file for PMCs that require a log file. The **hwpmc** driver will write log data to this file asynchronously. If it encounters an error, logging will be stopped and the error code encountered will be saved for subsequent retrieval by a PMC_OP_FLUSHLOG request.

PMC_OP_FLUSHLOG
: Transfer buffered log data inside **hwpmc** to a configured output file. This operation returns to the caller after the write operation has returned. The returned error code reflects any pending error state inside **hwpmc**.

PMC_OP_GETCPUINFO
: Retrieve information about the highest possible CPU number for the system, and the number of hardware performance monitoring counters available per CPU.

PMC_OP_GETDRIVERSTATS
: Retrieve module statistics (for analyzing the behavior of **hwpmc** itself).

PMC_OP_GETMODULEVERSION
: Retrieve the version number of API.

PMC_OP_GETPMCINFO
: Retrieve information about the current state of the PMCs on a given CPU.

PMC_OP_PMCADMIN
: Set the administrative state (i.e., whether enabled or disabled) for the hardware PMCs managed by the **hwpmc** driver. The invoking process needs to possess the PRIV_PMC_MANAGE privilege.

PMC_OP_PMCALLOCATE
: Allocate and configure a PMC. On successful allocation, a handle to the PMC (a 32 bit value) is returned.

PMC_OP_PMCATTACH
: Attach a process mode PMC to a target process. The PMC will be active whenever a thread in the target process is scheduled on a CPU.

  If the PMC_F_DESCENDANTS flag had been specified at PMC allocation time, then the PMC is attached to all current and future descendants of the target process.

PMC_OP_PMCDETACH
: Detach a PMC from its target process.

PMC_OP_PMCRELEASE
: Release a PMC.

PMC_OP_PMCRW
: Read and write a PMC. This operation is valid only for PMCs configured in counting modes.

PMC_OP_SETCOUNT
: Set the initial count (for counting mode PMCs) or the desired sampling rate (for sampling mode PMCs).

PMC_OP_PMCSTART
: Start a PMC.

PMC_OP_PMCSTOP
: Stop a PMC.

PMC_OP_WRITELOG
: Insert a timestamped user record into the log file.

i386 Specific API

Some i386 family CPUs support the RDPMC instruction which allows a user process to read a PMC value without needing to invoke a PMC_OP_PMCRW operation. On such CPUs, the machine address associated with an allocated PMC is retrievable using the PMC_OP_PMCX86GETMSR system call.

PMC_OP_PMCX86GETMSR
: Retrieve the MSR (machine specific register) number associated with the given PMC handle.

The PMC needs to be in process-private mode and allocated without the PMC_F_DESCENDANTS modifier flag, and should be attached only to its owner process at the time of the call.

amd64 Specific API

AMD64 CPUs support the RDPMC instruction which allows a user process to read a PMC value without needing to invoke a PMC_OP_PMCRW operation. The machine address associated with an allocated PMC is retrievable using the PMC_OP_PMCX86GETMSR system call.

PMC_OP_PMCX86GETMSR

Retrieve the MSR (machine specific register) number associated with the given PMC handle.

The PMC needs to be in process-private mode and allocated without the PMC_F_DESCENDANTS modifier flag, and should be attached only to its owner process at the time of the call.

SYSCTL VARIABLES AND LOADER TUNABLES

The behavior of **hwpmc** is influenced by the following sysctl(8) and loader(8) tunables:

kern.hwpmc.callchaindepth (integer, read-only)

The maximum number of call chain records to capture per sample. The default is 8.

kern.hwpmc.debugflags (string, read-write)

(Only available if the **hwpmc** driver was compiled with **-DDEBUG**.) Control the verbosity of debug messages from the **hwpmc** driver.

kern.hwpmc.hashsize (integer, read-only)

The number of rows in the hash tables used to keep track of owner and target processes. The default is 16.

kern.hwpmc.logbuffersize (integer, read-only)

The size in kilobytes of each log buffer used by **hwpmc**'s logging function. The default buffer size is 4KB.

kern.hwpmc.mincount (integer, read-write)

The minimum sampling rate for sampling mode PMCs. The default count is 1000 events.

kern.hwpmc.mtxpoolsize (integer, read-only)

The size of the spin mutex pool used by the PMC driver. The default is 32.

kern.hwpmc.nbuffers_pcpu (integer, read-only)
The number of log buffers used by **hwpmc** for logging. The default is 64.

kern.hwpmc.nsamples (integer, read-only)
The number of entries in the per-CPU ring buffer used during sampling. The default is 512.

security.bsd.unprivileged_syspmcs (boolean, read-write)
If set to non-zero, allow unprivileged processes to allocate system-wide PMCs. The default value is 0.

security.bsd.unprivileged_proc_debug (boolean, read-write)
If set to 0, the **hwpmc** driver will only allow privileged processes to attach PMCs to other processes.

These variables may be set in the kernel environment using kenv(1) before **hwpmc** is loaded.

IMPLEMENTATION NOTES

SMP Symmetry

The kernel driver requires all physical CPUs in an SMP system to have identical performance monitoring counter hardware.

Sparse CPU Numbering

On platforms that sparsely number CPUs and which support hot-plugging of CPUs, requests that specify non-existent or disabled CPUs will fail with an error. Applications allocating system-scope PMCs need to be aware of the possibility of such transient failures.

x86 TSC Handling

Historically, on the x86 architecture, FreeBSD has permitted user processes running at a processor CPL of 3 to read the TSC using the RDTSC instruction. The **hwpmc** driver preserves this behavior.

Intel P4/HTT Handling

On CPUs with HTT support, Intel P4 PMCs are capable of qualifying only a subset of hardware events on a per-logical CPU basis. Consequently, if HTT is enabled on a system with Intel Pentium P4 PMCs, then the **hwpmc** driver will reject allocation requests for process-private PMCs that request counting of hardware events that cannot be counted separately for each logical CPU.

DIAGNOSTICS

hwpmc: [class/npmc/capabilities]... Announce the presence of *npmc* PMCs of class *class*, with capabilities described by bit string *capabilities*.

hwpmc: kernel version (0x%x) does not match module version (0x%x). The module loading process failed because a version mismatch was detected between the currently executing kernel and the module being loaded.

hwpmc: this kernel has not been compiled with 'options HWPMC_HOOKS'. The module loading process failed because the currently executing kernel was not configured with the required configuration option HWPMC_HOOKS.

hwpmc: tunable hashsize=%d must be greater than zero. A negative value was supplied for tunable *kern.hwpmc.hashsize*.

hwpmc: tunable logbuffersize=%d must be greater than zero. A negative value was supplied for tunable *kern.hwpmc.logbuffersize*.

hwpmc: tunable nlogbuffers=%d must be greater than zero. A negative value was supplied for tunable *kern.hwpmc.nlogbuffers*.

hwpmc: tunable nsamples=%d out of range. The value for tunable *kern.hwpmc.nsamples* was negative or greater than 65535.

DEBUGGING

The **hwpmc** module can be configured to record trace entries using the ktr(4) interface. This is useful for debugging the driver's functionality, primarily during development. This debugging functionality is not enabled by default, and requires recompiling the kernel and **hwpmc** module after adding the following to the kernel config:

```
options KTR
options KTR_COMPILE=(KTR_SUBSYS)
options KTR_MASK=(KTR_SUBSYS)
options HWPMC_DEBUG
```

This alone is not enough to enable tracing; one must also configure the *kern.hwpmc.debugflags* sysctl(8) variable, which provides fine-grained control over which types of events are logged to the trace buffer.

hwpmc trace events are grouped by 'major' and 'minor' flag types. The major flag names are as follows:

cpu	CPU events
csw	Context switch events
logging	Logging events

md	Machine-dependent/class-dependent events
module	Miscellaneous events
owner	PMC owner events
pmc	PMC management events
process	Process events
sampling	Sampling events

The minor flags for each major flag group can vary. The individual minor flag names are:

allocaterow, allocate, attach, bind, config, exec, exit, find, flush, fork, getbuf, hook, init, intr, linktarget, mayberemove, ops, read, register, release, remove, sample, scheduleio, select, signal, swi, swo, start, stop, syscall, unlinktarget, write

The *kern.hwpmc.debugflags* variable is a string with a custom format. The string should contain a space-separated list of event specifiers. Each event specifier consists of the major flag name, followed by an equal sign (=), followed by a comma-separated list of minor event types. To track all events for a major group, an asterisk (*) can be given instead of minor event names.

For example, to trace all allocation and release events, set *debugflags* as follows:

```
kern.hwpmc.debugflags="pmc=allocate,release md=allocate,release"
```

To trace all events in the process and context switch major flag groups:

```
kern.hwpmc.debugflags="process=* csw=*"
```

To disable all trace events, set the variable to an empty string.

```
kern.hwpmc.debugflags=""
```

Trace events are recorded by ktr(4), and can be inspected at run-time using the ktrdump(8) utility, or at the ddb(4) prompt after a panic with the ’show ktr’ command.

ERRORS

A command issued to the **hwpmc** driver may fail with the following errors:

[EAGAIN]	Helper process creation failed for a PMC_OP_CONFIGURELOG request due to a temporary resource shortage in the kernel.
[EBUSY]	A PMC_OP_CONFIGURELOG operation was requested while an existing log

was active.

[EBUSY] A DISABLE operation was requested using the PMC_OP_PMCADMIN request for a set of hardware resources currently in use for process-private PMCs.

[EBUSY] A PMC_OP_PMCADMIN operation was requested on an active system mode PMC.

[EBUSY] A PMC_OP_PMCATTACH operation was requested for a target process that already had another PMC using the same hardware resources attached to it.

[EBUSY] A PMC_OP_PMCRW request writing a new value was issued on a PMC that was active.

[EBUSY] A PMC_OP_PMCSETCOUNT request was issued on a PMC that was active.

[EDOOFUS] A PMC_OP_PMCSTART operation was requested without a log file being configured for a PMC allocated with PMC_F_LOG_PROCCSW and PMC_F_LOG_PROCEXIT modifiers.

[EDOOFUS] A PMC_OP_PMCSTART operation was requested on a system-wide sampling PMC without a log file being configured.

[EEXIST] A PMC_OP_PMCATTACH request was reissued for a target process that already is the target of this PMC.

[EFAULT] A bad address was passed in to the driver.

[EINVAL] An invalid PMC handle was specified.

[EINVAL] An invalid CPU number was passed in for a PMC_OP_GETPMCINFO operation.

[EINVAL] The *pm_flags* argument to a PMC_OP_CONFIGURELOG request contained unknown flags.

[EINVAL] A PMC_OP_CONFIGURELOG request to de-configure a log file was issued without a log file being configured.

[EINVAL] A PMC_OP_FLUSHLOG request was issued without a log file being configured.

[EINVAL] An invalid CPU number was passed in for a PMC_OP_PMCADMIN operation.

[EINVAL] An invalid operation request was passed in for a PMC_OP_PMCADMIN operation.

[EINVAL] An invalid PMC ID was passed in for a PMC_OP_PMCADMIN operation.

[EINVAL] A suitable PMC matching the parameters passed in to a PMC_OP_PMCALLOCATE request could not be allocated.

[EINVAL] An invalid PMC mode was requested during a PMC_OP_PMCALLOCATE request.

[EINVAL] An invalid CPU number was specified during a PMC_OP_PMCALLOCATE request.

[EINVAL] A CPU other than PMC_CPU_ANY was specified in a PMC_OP_PMCALLOCATE request for a process-private PMC.

[EINVAL] A CPU number of PMC_CPU_ANY was specified in a PMC_OP_PMCALLOCATE request for a system-wide PMC.

[EINVAL] The *pm_flags* argument to an PMC_OP_PMCALLOCATE request contained unknown flags.

[EINVAL] (On Intel Pentium 4 CPUs with HTT support) A PMC_OP_PMCALLOCATE request for a process-private PMC was issued for an event that does not support counting on a per-logical CPU basis.

[EINVAL] A PMC allocated for system-wide operation was specified with a PMC_OP_PMCATTACH or PMC_OP_PMCDETACH request.

[EINVAL] The *pm_pid* argument to a PMC_OP_PMCATTACH or PMC_OP_PMCDETACH request specified an illegal process ID.

[EINVAL] A PMC_OP_PMCDETACH request was issued for a PMC not attached to the target process.

[EINVAL] Argument *pm_flags* to a PMC_OP_PMCRW request contained illegal flags.

[EINVAL] A PMC_OP_PMCX86GETMSR operation was requested for a PMC not in process-virtual mode, or for a PMC that is not solely attached to its owner process, or for a PMC that was allocated with flag PMC_F_DESCENDANTS.

[EINVAL] A PMC_OP_WRITELOG request was issued for an owner process without a log file configured.

[ENOMEM] The system was not able to allocate kernel memory.

[ENOSYS] (On i386 and amd64 architectures) A PMC_OP_PMCX86GETMSR operation was requested for hardware that does not support reading PMCs directly with the RDPMC instruction.

[ENXIO] A PMC_OP_GETPMCINFO operation was requested for an absent or disabled CPU.

[ENXIO] A PMC_OP_PMCALLOCATE operation specified allocation of a system-wide PMC on an absent or disabled CPU.

[ENXIO] A PMC_OP_PMCSTART or PMC_OP_PMCSTOP request was issued for a system-wide PMC that was allocated on a CPU that is currently absent or disabled.

[EOPNOTSUPP] A PMC_OP_PMCALLOCATE request was issued for PMC capabilities not supported by the specified PMC class.

[EOPNOTSUPP] (i386 architectures) A sampling mode PMC was requested on a CPU lacking an APIC.

[EPERM] A PMC_OP_PMCADMIN request was issued by a process without super-user privilege or by a jailed super-user process.

[EPERM] A PMC_OP_PMCATTACH operation was issued for a target process that the current process does not have permission to attach to.

[EPERM] (i386 and amd64 architectures) A PMC_OP_PMCATTACH operation was issued on a PMC whose MSR has been retrieved using PMC_OP_PMCX86GETMSR.

[ESRCH] A process issued a PMC operation request without having allocated any PMCs.

[ESRCH] A process issued a PMC operation request after the PMC was detached from all of its target processes.

[ESRCH] A PMC_OP_PMCATTACH or PMC_OP_PMCDETACH request specified a non-existent process ID.

[ESRCH] The target process for a PMC_OP_PMCDETACH operation is not being monitored by **hwpmc**.

SEE ALSO

kenv(1), pmc(3), pmclog(3), ddb(4), ktr(4), kldload(8), ktrdump(8), pmccontrol(8), pmcstat(8), sysctl(8), kproc_create(9), p_candebug(9)

HISTORY

The **hwpmc** driver first appeared in FreeBSD 6.0.

AUTHORS

The **hwpmc** driver was written by Joseph Koshy <*jkoshy@FreeBSD.org*>.

BUGS

The driver samples the state of the kernel's logical processor support at the time of initialization (i.e., at module load time). On CPUs supporting logical processors, the driver could misbehave if logical processors are subsequently enabled or disabled while the driver is active.

On the i386 architecture, the driver requires that the local APIC on the CPU be enabled for sampling mode to be supported. Many single-processor motherboards keep the APIC disabled in BIOS; on such systems **hwpmc** will not support sampling PMCs.

SECURITY CONSIDERATIONS

PMCs may be used to monitor the actual behavior of the system on hardware. In situations where this constitutes an undesirable information leak, the following options are available:

1. Set the sysctl(8) tunable *security.bsd.unprivileged_syspmcs* to 0. This ensures that unprivileged processes cannot allocate system-wide PMCs and thus cannot observe the hardware behavior of the system as a whole. This tunable may also be set at boot time using loader(8), or with kenv(1) prior to loading the **hwpmc** driver into the kernel.

2. Set the sysctl(8) tunable *security.bsd.unprivileged_proc_debug* to 0. This will ensure that an unprivileged process cannot attach a PMC to any process other than itself and thus cannot observe the hardware behavior of other processes with the same credentials.

System administrators should note that on IA-32 platforms FreeBSD makes the content of the IA-32 TSC counter available to all processes via the RDTSC instruction.