NAME

pmc - library for accessing hardware performance monitoring counters

LIBRARY

Performance Counters Library (libpmc, -lpmc)

SYNOPSIS

#include <pmc.h>

DESCRIPTION

The Performance Counters Library (libpmc, -lpmc) provides a programming interface that allows applications to use hardware performance counters to gather performance data about specific processes or for the system as a whole. The library is implemented using the lower-level facilities offered by the hwpmc(4) driver.

Key Concepts

Performance monitoring counters (PMCs) are represented by the library using a software abstraction. These "abstract" PMCs can have two scopes:

- System scope. These PMCs measure events in a whole-system manner, i.e., independent of the currently executing thread. System scope PMCs are allocated on specific CPUs and do not migrate between CPUs. Non-privileged process are allowed to allocate system scope PMCs if the hwpmc(4) sysctl tunable: *security.bsd.unprivileged_syspmcs* is non-zero.

- Process scope. These PMCs only measure hardware events when the processes they are attached to are executing on a CPU. In an SMP system, process scope PMCs migrate between CPUs along with their target processes.

Orthogonal to PMC scope, PMCs may be allocated in one of two operational modes:

- Counting PMCs measure events according to their scope (system or process). The application needs to explicitly read these counters to retrieve their value.

- Sampling PMCs cause the CPU to be periodically interrupted and information about its state of execution to be collected. Sampling PMCs are used to profile specific processes and kernel threads or to profile the system as a whole.

The scope and operational mode for a software PMC are specified at PMC allocation time. An application is allowed to allocate multiple PMCs subject to availability of hardware resources.

The library uses human-readable strings to name the event being measured by hardware. The syntax used for specifying a hardware event along with additional event specific qualifiers (if any) is described in detail in section *EVENT SPECIFIERS* below.

PMCs are associated with the process that allocated them and will be automatically reclaimed by the system when the process exits. Additionally, process-scope PMCs have to be attached to one or more target processes before they can perform measurements. A process-scope PMC may be attached to those target processes that its owner process would otherwise be permitted to debug. An owner process may attach PMCs to itself allowing it to measure its own behavior. Additionally, on some machine architectures, such self-attached PMCs may be read cheaply using specialized instructions supported by the processor.

Certain kinds of PMCs require that a log file be configured before they may be started. These include:

- System scope sampling PMCs.
- Process scope sampling PMCs.
- Process scope counting PMCs that have been configured to report PMC readings on process context switches or process exits.

Up to one log file may be configured per owner process. Events logged to a log file may be subsequently analyzed using the pmclog(3) family of functions.

Supported CPUs

The CPUs known to the PMC library are named by the *enum pmc_cputype* enumeration. Supported CPUs include:

PMC_CPU_AMD_K7	AMD Athlon CPUs.
PMC_CPU_AMD_K8	AMD Athlon64 CPUs.
PMC_CPU_ARMV7_CORTEX_A5	ARMv7 Cortex A5 CPUs.
PMC_CPU_ARMV7_CORTEX_A7	ARMv7 Cortex A7 CPUs.
PMC_CPU_ARMV7_CORTEX_A8	ARMv7 Cortex A8 CPUs.
PMC_CPU_ARMV7_CORTEX_A9	ARMv7 Cortex A9 CPUs.
PMC_CPU_ARMV7_CORTEX_A15	ARMv7 Cortex A15 CPUs.
PMC_CPU_ARMV7_CORTEX_A17	ARMv7 Cortex A17 CPUs.
PMC_CPU_ARMV8_CORTEX_A53	ARMv8 Cortex A53 CPUs.
PMC_CPU_ARMV8_CORTEX_A57	ARMv8 Cortex A57 CPUs.
PMC_CPU_ARMV8_CORTEX_A76	ARMv8 Cortex A76 CPUs.
GENERIC	Generic

PMC_CPU_INTEL_ATOM	Intel Atom CPUs and other CPUs conforming to version 3 of the Intel performance measurement architecture.
PMC_CPU_INTEL_CORE	Intel Core Solo and Core Duo CPUs, and other CPUs conforming to version 1 of the Intel performance measurement architecture.
PMC_CPU_INTEL_CORE2	Intel Core2 Solo, Core2 Duo and Core2 Extreme CPUs, and other CPUs conforming to version 2 of the Intel performance measurement architecture.
PMC_CPU_PPC_7450	PowerPC MPC7450 CPUs.
PMC_CPU_PPC_970	IBM PowerPC 970 CPUs.
PMC_CPU_PPC_E500	PowerPC e500 Core CPUs.
PMC_CPU_PPC_POWER8	IBM POWER8 and POWER9 CPUs.

Supported PMCs

PMCs supported by this library are named by the *enum pmc_class* enumeration. Supported PMC classes include:

PMC_CLASS_IAF	Fixed function hardware counters presents in CPUs conforming to the Intel performance measurement architecture version 2 and later.
PMC_CLASS_IAP	Programmable hardware counters present in CPUs conforming to the Intel performance measurement architecture version 1 and later.
PMC_CLASS_K7	Programmable hardware counters present in AMD Athlon CPUs.
PMC_CLASS_K8	Programmable hardware counters present in AMD Athlon64 CPUs.
PMC_CLASS_TSC	The timestamp counter on i386 and amd64 architecture CPUs.
PMC_CLASS_ARMV7	ARMv7
PMC_CLASS_ARMV8	ARMv8
PMC_CLASS_PPC970	IBM PowerPC 970 class.
PMC_CLASS_POWER8	IBM POWER8 class.
PMC_CLASS_SOFT	Software events.

PMC Capabilities

Capabilities of performance monitoring hardware are denoted using the *enum pmc_caps* enumeration. Supported capabilities include:

PMC_CAP_CASCADE	The ability to cascade counters.
PMC_CAP_DOMWIDE	Separate counters tied to each NUMA domain.
PMC_CAP_EDGE	The ability to count negated to asserted transitions of the hardware conditions being probed for.
PMC_CAP_INTERRUPT	The ability to interrupt the CPU.
PMC_CAP_INVERT	The ability to invert the sense of the hardware conditions being measured.

PMC_CAP_PRECISE	The ability to perform precise sampling.
PMC_CAP_QUALIFIER	The hardware allows monitored to be further qualified in some system dependent way.
PMC_CAP_READ	The ability to read from performance counters.
PMC_CAP_SYSTEM	The ability to restrict counting of hardware events to when the CPU is running privileged code.
PMC_CAP_SYSWIDE	A single counter aggregating events for the whole system.
PMC_CAP_THRESHOLD	The ability to ignore simultaneous hardware events below a programmable threshold.
PMC_CAP_USER	The ability to restrict counting of hardware events to those when the CPU is running unprivileged code.
PMC_CAP_WRITE	The ability to write to performance counters.

CPU Naming Conventions

CPUs are named using small integers from zero up to, but excluding, the value returned by function **pmc_ncpu**(). On platforms supporting sparsely numbered CPUs not all the numbers in this range will denote valid CPUs. Operations on non-existent CPUs will return an error.

Functional Grouping of the API

This section contains a brief overview of the available functionality in the PMC library. Each function listed here is described further in its own manual page.

Administration

pmc_disable(), **pmc_enable**()
: Administratively disable (enable) specific performance monitoring counter hardware. Counters that are disabled will not be available to applications to use.

Convenience Functions

pmc_event_names_of_class()
: Returns a list of event names supported by a given PMC type.

pmc_name_of_capability()
: Convert a PMC_CAP_* flag to a human-readable string.

pmc_name_of_class()
: Convert a PMC_CLASS_* constant to a human-readable string.

pmc_name_of_cputype()
: Return a human-readable name for a CPU type.

pmc_name_of_disposition()
: Return a human-readable string describing a PMC’s disposition.

pmc_name_of_event()
: Convert a numeric event code to a human-readable string.

pmc_name_of_mode()
: Convert a PMC_MODE_* constant to a human-readable name.

pmc_name_of_state()
: Return a human-readable string describing a PMC's current state.

Library Initialization

pmc_init()
: Initialize the library. This function must be called before any other library function.

Log File Handling

pmc_configure_logfile()
: Configure a log file for hwpmc(4) to write logged events to.

pmc_flush_logfile()
: Flush all pending log data in hwpmc(4)'s buffers.

pmc_close_logfile()
: Flush all pending log data and close hwpmc(4)'s side of the stream.

pmc_writelog()
: Append arbitrary user data to the current log file.

PMC Management

pmc_allocate(), **pmc_release**()
: Allocate (free) a PMC.

pmc_attach(), **pmc_detach**()
: Attach (detach) a process scope PMC to a target.

pmc_read(), **pmc_write**(), **pmc_rw**()
: Read (write) a value from (to) a PMC.

pmc_start(), **pmc_stop**()
: Start (stop) a software PMC.

pmc_set()
: Set the reload value for a sampling PMC.

Queries

pmc_capabilities()
: Retrieve the capabilities for a given PMC.

pmc_cpuinfo()
: Retrieve information about the CPUs and PMC hardware present in the system.

pmc_get_driver_stats()
: Retrieve statistics maintained by hwpmc(4).

pmc_ncpu()
: Determine the greatest possible CPU number on the system.

pmc_npmc()
Return the number of hardware PMCs present in a given CPU.

pmc_pmcinfo()
Return information about the state of a given CPU's PMCs.

pmc_width()
Determine the width of a hardware counter in bits.

x86 Architecture Specific API

pmc_get_msr()
Returns the processor model specific register number associated with *pmc*. Applications may then use the x86 **RDPMC** instruction to directly read the contents of the PMC.

Signal Handling Requirements

Applications using PMCs are required to handle the following signals:

SIGBUS When the hwpmc(4) module is unloaded using kldunload(8), processes that have PMCs allocated to them will be sent a SIGBUS signal.

SIGIO The hwpmc(4) driver will send a PMC owning process a SIGIO signal if:

- any process-mode PMC allocated by it loses all its target processes.
- the driver encounters an error when writing log data to a configured log file. This error may be retrieved by a subsequent call to **pmc_flush_logfile**().

Typical Program Flow

1. An application would first invoke function **pmc_init**() to allow the library to initialize itself.
2. Signal handling would then be set up.
3. Next the application would allocate the PMCs it desires using function **pmc_allocate**().
4. Initial values for PMCs may be set using function **pmc_set**().
5. If a log file is necessary for the PMCs to work, it would be configured using function **pmc_configure_logfile**().
6. Process scope PMCs would then be attached to their target processes using function **pmc_attach**().
7. The PMCs would then be started using function **pmc_start**().

8. Once started, the values of counting PMCs may be read using function **pmc_read**(). For PMCs that write events to the log file, this logged data would be read and parsed using the pmclog(3) family of functions.

9. PMCs are stopped using function **pmc_stop**(), and process scope PMCs are detached from their targets using function **pmc_detach**().

10. Before the process exits, it may release its PMCs using function **pmc_release**(). Any configured log file may be closed using function **pmc_configure_logfile**().

EVENT SPECIFIERS

Event specifiers are strings comprising of an event name, followed by optional parameters modifying the semantics of the hardware event being probed. Event names are PMC architecture dependent, but the PMC library defines machine independent aliases for commonly used events.

Event specifiers spellings are case-insensitive and space characters, periods, underscores and hyphens are considered equivalent to each other. Thus the event specifiers "Example Event", "example-event", and "EXAMPLE_EVENT" are equivalent.

PMC Architecture Dependent Events

PMC architecture dependent event specifiers are described in the following manual pages:

PMC Class	*Manual Page*
PMC_CLASS_IAF	pmc.iaf(3)
PMC_CLASS_IAP	pmc.atom(3), pmc.core(3), pmc.core2(3)
PMC_CLASS_K7	pmc.k7(3)
PMC_CLASS_K8	pmc.k8(3)
PMC_CLASS_TSC	pmc.tsc(3)

Event Name Aliases

Event name aliases are PMC-independent names for commonly used events. The following aliases are known to this version of the **pmc** library:

branches
: Measure the number of branches retired.

branch-mispredicts
: Measure the number of retired branches that were mispredicted.

cycles Measure processor cycles. This event is implemented using the processor’s Time Stamp Counter

register.

dc-misses
: Measure the number of data cache misses.

ic-misses
: Measure the number of instruction cache misses.

instructions
: Measure the number of instructions retired.

interrupts
: Measure the number of interrupts seen.

unhalted-cycles
: Measure the number of cycles the processor is not in a halted or sleep state.

COMPATIBILITY

The interface between the **pmc** library and the hwpmc(4) driver is intended to be private to the implementation and may change. In order to ease forward compatibility with future versions of the hwpmc(4) driver, applications are urged to dynamically link with the **pmc** library. Doing otherwise is unsupported.

SEE ALSO

pmc.atom(3), pmc.core(3), pmc.core2(3), pmc.haswell(3), pmc.haswelluc(3), pmc.haswellxeon(3), pmc.iaf(3), pmc.ivybridge(3), pmc.ivybridgexeon(3), pmc.k7(3), pmc.k8(3), pmc.sandybridge(3), pmc.sandybridgeuc(3), pmc.sandybridgexeon(3), pmc.soft(3), pmc.tsc(3), pmc.westmere(3), pmc.westmereuc(3), pmc_allocate(3), pmc_attach(3), pmc_capabilities(3), pmc_configure_logfile(3), pmc_disable(3), pmc_event_names_of_class(3), pmc_get_driver_stats(3), pmc_get_msr(3), pmc_init(3), pmc_name_of_capability(3), pmc_read(3), pmc_set(3), pmc_start(3), pmclog(3), hwpmc(4), pmccontrol(8), pmcstat(8)

HISTORY

The **pmc** library first appeared in FreeBSD 6.0.

AUTHORS

The Performance Counters Library (libpmc, -lpmc) library was written by Joseph Koshy *<jkoshy@FreeBSD.org>*.