

NAME

pmclog_open, **pmclog_close**, **pmclog_read**, **pmclog_feed** - parse event log data generated by hwpmc(4)

LIBRARY

Performance Counters Library (libpmc, -lpmc)

SYNOPSIS

```
#include <pmclog.h>
```

```
void *
```

```
pmclog_open(int fd);
```

```
void
```

```
pmclog_close(void *cookie);
```

```
int
```

```
pmclog_read(void *cookie, struct pmclog_ev *ev);
```

```
int
```

```
pmclog_feed(void *cookie, char *data, int len);
```

DESCRIPTION

These functions provide a way for application programs to extract events from an event stream generated by hwpmc(4).

A new event log parser is allocated using **pmclog_open**(*fd*). Argument *fd* may be a file descriptor opened for reading if the event stream is present in a file, or the constant PMCLOG_FD_NONE for an event stream present in memory. This function returns a cookie that is passed into the other functions in this API set.

Function **pmclog_read**(*cookie*) returns the next available event in the event stream associated with argument *cookie*. Argument *ev* points to an event descriptor that which will contain the result of a successfully parsed event.

An event descriptor returned by **pmclog_read**(*cookie*) has the following structure:

```
struct pmclog_ev {
    enum pmclog_state pl_state;    /* parser state after 'get_event()' */
    off_t             pl_offset;   /* byte offset in stream */
    size_t            pl_count;   /* count of records so far */
};
```

```

struct timespec  pl_ts;          /* log entry timestamp */
enum pmclog_type pl_type;      /* log entry kind */
union {
    struct pmclog_ev_callchain pl_cc;
    struct pmclog_ev_closetlog pl_cl;
    struct pmclog_ev_dropnotify pl_d;
    struct pmclog_ev_initialize pl_i;
    struct pmclog_ev_map_in    pl_mi;
    struct pmclog_ev_map_out   pl_mo;
    struct pmclog_ev_pmcallocate pl_a;
    struct pmclog_ev_pmcallocatedyn pl_ad;
    struct pmclog_ev_pmcattach pl_t;
    struct pmclog_ev_pmcdetach pl_d;
    struct pmclog_ev_proccsw   pl_c;
    struct pmclog_ev_procexec  pl_x;
    struct pmclog_ev_procexit  pl_e;
    struct pmclog_ev_procfork  pl_f;
    struct pmclog_ev_sysexit   pl_e;
    struct pmclog_ev_userdata  pl_u;
} pl_u;
};

```

The current state of the parser is recorded in *pl_state*. This field can take on the following values:

PMCLOG_EOF	(For file based parsers only) An end-of-file condition was encountered on the configured file descriptor.
PMCLOG_ERROR	An error occurred during parsing.
PMCLOG_OK	A complete event record was read into <i>*ev</i> .
PMCLOG_REQUIRE_DATA	There was insufficient data in the event stream to assemble a complete event record. For memory based parsers, more data can be fed to the parser using function pmclog_feed() . For file based parsers, function pmclog_read() may be retried when data is available on the configured file descriptor.

The rest of the event structure is valid only if field *pl_state* contains PMCLOG_OK. Field *pl_offset* contains the offset of the current record in the byte stream. Field *pl_count* contains the serial number of this event. Field *pl_ts* contains a timestamp with the system time when the event occurred. Field

pl_type denotes the kind of the event returned in argument **ev* and is one of the following:

PMCLOG_TYPE_CLOSELOG	A marker indicating a successful close of a log file. This record will be the last record of a log file.
PMCLOG_TYPE_DROPNOTIFY	A marker indicating that <code>hwpmc(4)</code> had to drop data due to a resource constraint.
PMCLOG_TYPE_INITIALIZE	An initialization record. This is the first record in a log file.
PMCLOG_TYPE_MAP_IN	A record describing the introduction of a mapping to an executable object by a <code>kldload(2)</code> or <code>mmap(2)</code> system call.
PMCLOG_TYPE_MAP_OUT	A record describing the removal of a mapping to an executable object by a <code>kldunload(2)</code> or <code>munmap(2)</code> system call.
PMCLOG_TYPE_PCSAMPLE	A record containing an instruction pointer sample.
PMCLOG_TYPE_PMCALLOCATE	A record describing a PMC allocation operation.
PMCLOG_TYPE_PMCATTACH	A record describing a PMC attach operation.
PMCLOG_TYPE_PMCDETACH	A record describing a PMC detach operation.
PMCLOG_TYPE_PROCCSW	A record describing a PMC reading at the time of a process context switch.
PMCLOG_TYPE_PROCEXEC	A record describing an <code>execve(2)</code> by a target process.
PMCLOG_TYPE_PROCEXIT	A record describing the accumulated PMC reading for a process at the time of <code>_exit(2)</code> .
PMCLOG_TYPE_PROCFORK	A record describing a <code>fork(2)</code> by a target process.
PMCLOG_TYPE_SYSEXIT	A record describing a process exit, sent to processes owning system-wide sampling PMCs.
PMCLOG_TYPE_USERDATA	A record containing user data.

Function **`pmclog_feed()`** is used with parsers configured to parse memory based event streams. It is

intended to be called when function **pmclog_read()** indicates the need for more data by a returning `PMCLOG_REQUIRE_DATA` in field `pl_state` of its event structure argument. Argument `data` points to the start of a memory buffer containing fresh event data. Argument `len` indicates the number of data bytes available. The memory range `[data, data + len]` must remain valid till the next time **pmclog_read()** returns an error. It is an error to use **pmclog_feed()** on a parser configured to parse file data.

Function **pmclog_close()** releases the internal state allocated by a prior call to **pmclog_open()**.

RETURN VALUES

Function **pmclog_open()** will return a non-NULL value if successful or NULL otherwise.

Function **pmclog_read()** will return 0 in case a complete event record was successfully read, or will return -1 and will set the `pl_state` field of the event record to the appropriate code in case of an error.

Function **pmclog_feed()** will return 0 on success or -1 in case of failure.

EXAMPLES

A template for using the log file parsing API is shown below in pseudocode:

```
void *parser;                /* cookie */
struct pmclog_ev ev;        /* parsed event */
int fd;                     /* file descriptor */

fd = open(filename, O_RDONLY); /* open log file */
parser = pmclog_open(fd); /* initialize parser */
if (parser == NULL)
    --handle an out of memory error--;

/* read and parse data */
while (pmclog_read(parser, &ev) == 0) {
    assert(ev.pl_state == PMCLOG_OK);
    /* process the event */
    switch (ev.pl_type) {
    case PMCLOG_TYPE_ALLOCATE:
        --process a pmc allocation record--
        break;
    case PMCLOG_TYPE_PROCCSW:
        --process a thread context switch record--
        break;
    case PMCLOG_TYPE_CALLCHAIN:
```

```
                --process a callchain sample--
                break;
            --and so on--
        }
    }

    /* examine parser state */
    switch (ev.pl_state) {
    case PMCLOG_EOF:
        --normal termination--
        break;
    case PMCLOG_ERROR:
        --look at errno here--
        break;
    case PMCLOG_REQUIRE_DATA:
        --arrange for more data to be available for parsing--
        break;
    default:
        assert(0);
        /*NOTREACHED*/
    }

    pmclog_close(parser);                /* cleanup */
```

ERRORS

A call to **pmclog_init_parser()** may fail with any of the errors returned by **malloc(3)**.

A call to **pmclog_read()** for a file based parser may fail with any of the errors returned by **read(2)**.

SEE ALSO

read(2), **malloc(3)**, **pmc(3)**, **hwpmc(4)**, **pmcstat(8)**

HISTORY

The **pmclog** API first appeared in FreeBSD 6.0.