NAME

pcs − program counter sampling device

DESCRIPTION

Pcs provides an interface to program counter sampling, allowing for a statistical approach to user and kernel process profiling. *Pcs* is a read-only pseudo device supporting open, read, close, and ioctl functions. An open of *pcs* obtains exclusive use of the profiling device and starts the profiling clock. The profiling clock is assumed to be a TCU-100 (battery clock) or a KW11-K, if no TCU-100 clock exists. The clock should run at hardware and software priority 7. Thereafter, it is necessary to do an ioctl function to start gathering data. *Pcs* supports the following ioctl requests:

ioctl(fd, PROF_KERNEL, NULL)
: requests that sample points for the unix kernel be output. The third argument is unused.

ioctl(fd, PROF_ALL, NULL)
: requests that sample points be generated whenever any user or kernel process is interrupted. Again, the third argument is unused.

ioctl(fd, PROF_LIST, list)
: requests that sample points for the user level processes specified by the list argument be output. The list argument points to an array of integers; the first element is the number of processes to be sampled, and the remaining values are their process ids. The number of processes in the list is bounded by the number MAX_LIST in **pcs.h**.

ioctl(fd, PROF_GROUP, groupid)
: requests that sample points be output for all of the processes in process group *groupid*.

ioctl(fd, BUF_INCR, incr)
: requests that *incr* more system buffers be allocated for the collection of data. By default, DEF_BUFNO buffers are assigned to the device. *Incr* must be positive, and small enough so that no more than MAX_BUFS are be allocated to the device. (See **pcs.h** for default and max values).

Reads from *pcs* may be for an arbitrary number of bytes, although, in general, partial buffers are not made available to the user until filled with sample points. *Pcs* internally works in terms of standard UNIX buffers, BSIZE bytes in size. Such a buffer is described by the "LOGBUF" structure in **pcs.h**. It is a general layout, envisioned as being useful in reporting kernel and user generated "records" as well as miscellaneous and idle records. Basically, such a buffer consists of a header and several data records. A record is constrained to be no larger than one buffer (minus a buffer header), and in fact, a record is never split across buffers. For this reason, one entry in the header identifies the number of "unused" bytes at the end of the buffer. The unused count should be a small number (less than 12) for all blocks. Anticipating that the operating system will be able to generate records faster than a user process will be able consume them, the header also identifies the number of records "lost" since the last buffer was sent. For streams whose records are generated more slowly than the reading process' ability to consume them, this count should be 0. For high volume data, e.g. pc sampling at a very fast rate, or recording all type of hits on a busy system, this count may be non-zero, and although the data is lost, a count of lost data is provided. To further reduce the possibility of losing data, all system idle counts are stored internally and output every 100 clock cycles. The idle data is identified by sample type IDLE. The same technique is used to gather unasked-for kernel and user data. This data is stored internally and also output every 100 clock cycles as MKERNEL, MKERNELI or MUSER type of sample. If the profiling clock interrupted the processor when it was servicing an interrupt, this data will be output as KERNELI data or it may be stored

internally and output every 100 clock cycles as **MKERNELI** samples.

At the end of sampling, any data that remains in a system buffer is thrown away. This means that as many as three buffers worth of data may be lost when the *close* routine is called.

The data records generated by *pcs* are defined by the structure **PSAMPLE** or **MSAMPLE** in **pcs.h**. For **PSAMPLE** data, the *pid* field gives the process id of the interrupted process, and *pc* the value of its program counter. Also given is the type of sample, kernel or user, and the text space, meaningful only for kernel samples. The cpu interrupt priority level is included in the high 4 bits of *sspace* which again is only meaningful for kernel samples. **MSAMPLE** records are **MKERNEL, MKERNELI, MUSER** and **IDLE** data, *type* identifies the type of data in the sample and *count* is the actual number of hits that were recorded for this type of sample. The **pcs.h** header is as follows:

```
/*              @(#)pcs.h       3.1             */
/*
 *              These structures and macros are used by the SYSTEM PROFILING (pc)
 *              special character device, the data-gathering command getpc,
 *              and the analysis commands analpc.
 *
 *              The pc driver never generates records of types START, STOP, or IOCTL;
 *              these are created by getpc for the benefit of the analpc routines.
 *              The data stream presented by getpc will contain a START record, an
 *              IOCTL record, an arbitrary number of KERNEL, USER, MKERNEL, and
 *              MUSR records, and, finally, a STOP record.
 *              The IOCTL record is an indication of the ioctl system call made by
 *              the getpc program to the pc driver, indicating what data is available
 *              to analpc for reporting.
 */

/*              pc stream record types */

#define KERNEL          1
#define USER            2
#define IDLE            3
#define MKERNEL         4
#define MUSER           5
#define START           6
#define STOP            7
#define IOCTL           8
#define KERNELI         9
#define MKERNELI        10

#define DEF_BUFNO       3               /* default number of system buffers used */
#define MAX_BUFS        10              /* max buffers allowed to pc for profiling */
#define MAX_LIST        5               /* max number in process list for profiling */

struct PSAMPLE {                /* pc profile sample record */
        char            type;           /* KERNEL, KERNELI, or USER space */
        char            sspace;         /* Kernel switchable space number */
                                        /* high 4 bits have CPU priority level */
        short           pid;            /* pid of current user process */
        unsigned pc;    /* program counter */
};

struct MSAMPLE {                /* misc. or merged profile sample record */
        char            type;           /* MKERNEL, MKERNELI, MUSER, or IDLE hit count */
        char            cfill;          /* (structure pad) */
        short           count;          /* number of type of misc. records merged since
                                           last such count */
        unsigned ufill; /* (structure pad) */
};
```

```
struct SSAMPLE {                /* start or stop getpc-produced record */
        char            type;           /* START or STOP */
        char            cfill;          /* (structure pad) */
        time_t          stime;          /* start or stop time */
};

struct ISAMPLE {                /* ioctl record produced by getpc */
        char            type;           /* IOCTL */
        char            cfill;          /* (structure pad) */
        int             cmd;            /* pc ioctl call command */
        short           data[MAX_LIST];         /* ioctl arg., depends on cmd */
};

/*              pc ioctl commands */
#define BUF_INCR            (('P'<<8)|01)   /* incr. number of bufs for /dev/pc */
#define PROF_KERNEL         (('P'<<8)|02)   /* profile the kernel */
#define PROF_ALL            (('P'<<8)|04)   /* profile all user processes */
#define PROF_GROUP          (('P'<<8)|010)  /* profile a user group */
#define PROF_LIST           (('P'<<8)|020)  /* profile a small list of processes */
#define PROF_MASK           0177                    /* used to mask out high byte */

#define WAIT                1
#define NOWAIT              2

#define SAMPPRI             (PZERO+1)

#define NO_CLOCK            0
#define TCU100_CLOCK        1
#define KW11K_CLOCK         2

#define FILLING             1
#define NOT_FILLING         2

struct LOGBUF_HDR {                             /* pc buffer header info */
        ushort          h_unused;       /* num unused bytes at end of buf */
        short           h_numlost;      /* num samples lost between buffers */
};

#define NUMSAMPS   ((BSIZE-sizeof(struct LOGBUF_HDR))/sizeof(struct PSAMPLE))

#define NUMWASTE   (BSIZE-sizeof(struct LOGBUF_HDR)-(NUMSAMPS*sizeof(struct
                                        PSAMPLE)))

struct LOGBUF {                                 /* layout of pc system buffer */
        struct          LOGBUF_HDR lb_buf_hdr;      /* pc buffer header counts */
        struct          PSAMPLE lb_data[NUMSAMPS]; /* pc data samples */
        char            lb_wasted[NUMWASTE];        /* buffer bytes wasted */
};

struct samp_cntl {                              /* pc queue and buffer control info */
        ushort          flag;                           /* control flag */
        struct  buf *cursbuf;           /* current output buffer for sample data */
        caddr_t         currptr;        /* address in block for next sample */
        struct          buf *rq;        /* ready queue (filled for user to read) */
        struct          buf *fq;        /* free queue of empty buffers */
        struct          LOGBUF_HDR sbuf_hdr; /* unused buffer bytes; lost recs count */
};

struct pid_list {                               /* pids of processes to be profiled */
        short p_count;                  /* number in list */
        short p_list[MAX_LIST];         /* list of pids for profiling */
};
```

```
struct DEVPC {                                  /* pc pseudo-device pseudo-registers */
        short           d_clock;                /* type of profiling clock */
        short           d_runpid;               /* pid of user running pc device;
                                                        this is 0 if no one profiling */
        ushort          d_flag;                                 /* type of profiling being done */
        short           d_group;                /* group id profiling for */
        struct pid_list d_list; /* list of pids for profiling */
        struct samp_cntl d_smple; /* queue and buffer control info */
};
```

FILES

/dev/pcs

SEE ALSO

getpc(1), pcstat(1)