PSTAT Interfaces

May 2000

Table of Contents

1. Introduction

1.1 Abstract

Pstat is a supported Application Programming Interface (API) that provides many HP-UX system contexts. It provides a number of wrappers (pstat_get*) and corresponding structures (struct pst_*) to get information from the kernel. These interfaces are designed in such a way to allow future expansion of the interface, while preserving source and binary compatibility for programs written using the pstat interfaces. Pstat interfaces are available in both 64-bit and 32-bit versions.

Today, many applications read kernel data structures through unsupported interfaces, such as the /dev/kmem pseudo driver, to get information about open files, resource usage, process activity etc. Because kernel data structures change from release to release, this access method is fragile, incurring a high maintenance cost. Replacing such access methods with calls to the pstat() system call will insulate these applications from the release-to-release variability in private kernel data structures and eliminate the need to re-release these applications with each new HP-UX release.

1.2 Purpose of Document

The purpose of this document is to communicate the availability of pstat interfaces and to give developers guidance on how to take advantage of these supported interfaces. It provides code examples to demonstrate the calling conventions.

1.3 Intended Audience

This document is intended for HP-UX software developers, system administrators, and end users, both internal and external to Hewlett-Packard.

1.4 Structure of Document

The document is structured as follows. Section 2 provides an overview of pstat. Section 3 provides a summary of the system contexts available through pstat. Each of the pstat wrappers is described in detail in Section 4. Code examples that demonstrate the usage of pstat wrappers are also included. Section 5 explains how pstat supports binary compatibility. Each of the available contexts is explained in detail in Section 6.

1.5 Related Documentation

- Man page for pstat()
- Header files <sys/pstat.h> and <sys/pstat/*.h>

- For more information on how to use pstat interfaces please refer to the source code of freeware application *lsof* at ftp://vic.cc.purdue.edu/pub/tools/unix/lsof

2. Overview

In the early days of UNIX, programmers and system administrators were required to write their own tools to monitor and manage the system. In order to facilitate access to the internals of the kernel implementation, UNIX provided the /dev/kmem and /dev/mem pseudo devices. This was a reasonable thing to do in those days, as release-to-release binary compatibility was not an issue. Modern UNIX systems now provide the application programmers with a rich set of standard APIs. Writing applications that depend on kernel internals is no longer a recommended practice. In fact, exposure to kernel internals creates serious binary compatibility problems for user applications. The availability of many commercial applications for HP-UX makes the release-to-release binary compatibility a very important feature.

Even today many system management and measurement tools read kernel data structures through unsupported interfaces, such as /dev/kmem pseudo driver, to get information about open files, resource usage, process activity, etc. Because kernel data structures change from release to release, this access method is fragile, incurring a high maintenance cost. To insulate these applications from the release-to-release variability in private kernel data structures, HP-UX provides the pstat() system call and a set of wrappers. Pstat is a supported API that provides many of the HP-UX system contexts. It provides a number of wrappers (pstat_get*) and corresponding structures (struct pst_*) to get information from the kernel. These interfaces are designed in such a way to allow future expansion of the interface, while preserving source and binary compatibility for programs written using the pstat interfaces. Pstat interfaces are available in both 64-bit and 32-bit versions. Replacing the /dev/kmem access with pstat wrappers will eliminate the need to re-release these applications with each new HP-UX release.

3. Summary of Available Contexts

The general framework for pstat() interface is that the user programs call the libc wrappers with required parameters. The wrappers then call the pstat() system call, passing the required parameters. pstat() collects the required information and places it in the user's buffer. The pstat wrappers return information about various system contexts. The contents of the various associated data structures are described in the Appendix. The following contexts of information are returned by pstat wrappers: Static, Dynamic, Virtual Memory (VM), Inter Process Communication (IPC), Stable Store, Crash Dumps, Processor, Disk, Swap Areas, Dump Areas, Node (ccNUMA), Process, Light Weight Process (LWP), Process Region, Logical Volume Manager Volume, Semaphore Set, Message Queue, Shared Memory, Open File, Open Socket, Open Stream, and Dynamic Name Lookup Cache (DNLC). The following table summarizes the various contexts.

Context	Structure	Routine	Instances	Short Cut?
Static	pst_static	pstat_getstatic()	1	
Dynamic	pst_dynamic	pstat_getdynamic()	1	
VM	pst_vminfo	pstat_getvminfo	1	

IPC	pst_ipcinfo	pstat_getipc()	1	
Stable Store	pst_stable	pstat_getstable()	1	
Crash Dumps	pst_crashinfo	pstat_getcrashinfo()	1	
Processor	pst_processor	pstat_getprocessor()	1 per processor	
Disk	pst_diskinfo	pstat_getdisk()	1 per disk	
Swap Areas	pst_swapinfo	pstat_getswap()	1 per swap area	
Dump Areas	pst_crashdev	pstat_getcrashdev()	1 per dump area	
Node	pst_node	pstat_getnode()	1 per node	
Process	pst_status	pstat_getproc()	1 per process	yes
LW Process	lwp_status	pstat_getlwp()	1 per LWP/thread	yes
Process Region	pst_vm_status	pstat_getprocvm()	1 per process region	yes
LVM Volume	pst_lvinfo	pstat_getlv()	1 per logical volume	yes
Sema Set	pst_seminfo	pstat_getsem()	1 per semaphore set	yes
Msg Queue	pst_msginfo	pstat_getmsg()	1 per message queue	yes
Shared Memory	pst_shminfo	pstat_getshm()	1 per shm segment	yes
Open File	pst_fileinfo	pstat_getfile()	1 per open file/process	yes
Open File	pst_fileinfo2	pstat_getfile2()	1 per open file/process	yes
Open File	pst_filedetails	pstat_getfiledetails()	1 per open file/call	
Open Socket	pst_socket	pstat_getsocket()	1 per open socket/call	
Open Stream	pst_stream	pstat_getstream()	1 per open stream/call	
Open File	char *	pstat_getpathname()	1 per open file/call	
DNLC	pst_mpathnode	pstat_getmpathname()	1 per DNLC entry	

A wide (64-bit) version of the pstat interfaces is available for narrow (32-bit) applications to use. A narrow application could use the flag -D_PSTAT64 at compile time to switch to the wide interfaces. Using this compiler flag in a narrow application is equivalent to using the default interfaces on a wide system. The data structures defined in the respective header files have types such as _T_LONG_T. The reason for this is to have two different data structures, one for 64 bit and one for 32-bit. This is achieved as follows (from <sys/pstat.h>):

```
# if defined(_PSTAT64)
# define _T_ULONG_T uint64_t
# define _T_LONG_T int64_t
# else
# define _T_ULONG_T uint32_t
# define _T_LONG_T int32_t
# endif
```

For 32-bit application running on 64-bit kernels, it is possible that certain fields of the contexts may overflow. For example, the size of a file in pst_filedetails can overflow when large files are supported. To detect this situation, a valid vector is provided in all the data structures that require one. This vector will indicate if a certain field in the structure is valid or not. User should check this vector when EOVERFLOW is set. Macros for field validity are also provided. Initially, all the bits in the vector are set. These bits will be cleared in valid vector if the corresponding members overflow.

4. Description of Wrappers

4.1 pstat_getcrashdev()

Synopsis:
int pstat_getcrashdev(struct pst_crashdev *buf, size_t elemsize, size_t elemcount, int index);
Description:

This call returns information about crash dump devices configured on the system. There is one instance of this context for each crash dump device configured on the system. For each instance, data up to a maximum of elemsize bytes are returned in the structs pst_crashdev pointed to by buf. The elemcount parameter specifies the number of structs pst_crashdev that are available at buf. The index parameter specifies the starting index within the context of crash dump devices. This call is available only in wide mode. That is –D_PSTAT64 flag needs to be defined at compile time.

Return Value:

This call returns –1 on error or the number of instances copied to buf on success.

Errors:

- EINVAL is set if the user is in narrow more.
- EINVAL is set if elemcount is not >= 1 or index is not >= to 0.
- EINVAL is set if user's size declaration is not valid.
- EFAULT is set if buf is invalid.

Example:

```
#include <sys/pstat.h>

void main(void)
{
      struct pst_crashdev pst;
      int count;

      /* Get information about the first crash dump device configured. */
      count = pstat_getcrashdev(&pst, sizeof(pst), 1, 0);
      if (count > 0) {
            printf("\nSize of dump area: %d", pst.psc_size);
      } else {
            perror("pstat_getcrashdev()");
      }
}
```

4.2 pstat_getcrashinfo()

Synopsis:
int pstat_getcrashinfo(struct pst_crashinfo *buf, size_t elemsize, size_t elemcount, int index);
Description:

This call provides information about system's crash dump configuration. There is only one instance of this context. The elemcount parameter must be equal to 1 and the index parameter must be equal to 0. This call is available only in wide mode. That is –D_PSTAT64 flag needs to be defined at compile time.

Return Value:

This call returns –1 on failure or 1 on success.

Errors:

- EINVAL is set if the user is in narrow more.
- EINVAL is set if elemcount is not equal to 1 or index is not equal to 0.
- EINVAL is set if user's size declaration is not valid.
- EFAULT is set if buf is invalid.

Example:

```
#include <sys/pstat.h>

void main(void)
{
      struct pst_crashinfo pst;
      int count;

      /* Get information about the first crash dump configuration. */
      count = pstat_getcrashinfo(&pst, sizeof(pst), 1, 0);
      if (count > 0) {
            printf("\nSize of total dump area: %d", pst.psc_totalsize);
      } else {
            perror("pstat_getcrashinfo()");
      }
}
```

4.3 pstat_getdisk()

Synopsis:

int pstat_getdisk(struct pst_diskinfo *buf, size_t elemsize, size_t elemcount, int index);

Description:

This call returns information about disks configured in the system. There is one instance of this context for each disk configured. For each instance, data up to a maximum of elemsize bytes are returned in the structs pst_diskinfo pointed to by buf. The elemcount parameter specifies the number of structs pst_diskinfo that are available at buf. The index parameter specifies the starting index within the context of disks.

Return Value:

This call returns –1 on failure or number of instances copied to buf on success.

Errors:

- EINVAL is set if elemcount is not greater than or equal to 1 or index is not greater than or equal to 0.
- EINVAL is set if user's size declaration is not valid.
- EFAULT is set if buf is invalid.

Example:

```
#include <sys/pstat.h>

void main(void)
{
        struct pst_diskinfo pst[10];
        int i, count;

        /* Get information about disks configured in the system. */
        count = pstat_getdisk(pst, sizeof(struct pst_diskinfo),
                              sizeof(pst) / sizeof(struct pst_diskinfo), 0);
        if (count < 0) {
                perror("pstat_getdisk()");
                return;
```

```
        }
        for (i = 0; i < count; i++) {
                printf("\n Disk hardware path = %s",
                        pst[i].psd_hw_path.psh_name);
        }
}
```

4.4 pstat_getdynamic()

Synopsis:

int pstat_getdynamic(struct pst_dynamic *buf, size_t elemsize, size_t elemcount, int index);

Description:

This call returns dynamic system variables, ones that may change frequently during normal operation of the kernel. This includes such information, as the number of jobs in disk wait, free memory pages, open logical volumes, etc. There is only one instance of this context. Data up to a maximum of elemsize bytes are returned in the struct pst_dynamic pointed to by buf. The elemcount parameter must be 1. The index parameter must be 0.

Return Value:

This call returns –1 on failure or 1 on success.

Errors:

- EINVAL is set if elemcount is not equal to 1 or index is not equal to 0.
- EINVAL is set if user's size declaration is not valid.
- EFAULT is set if buf is invalid.

Example:

```
#include <sys/pstat.h>

void main(void)
{
        struct pst_dynamic pst;
        int count;

        /* Get information about the system dynamic variables */
        count = pstat_getdynamic(&pst, sizeof(pst), 1, 0);
        if (count == 1) {
                printf("\n Active processors: %d, DNLC size: %d",
                        pst.psd_proc_cnt, pst.psd_dnlc_size);
        } else {
                perror("pstat_getdynamic()");
        }
}
```

4.5 pstat_getfile() OBSOLETE

Synopsis:

int pstat_getfile(struct pst_fileinfo *buf, size_t elemsize, size_t elemcount, int index);

Description:

This call is now **obsolete** and is provided for backward compatibility. Use of the pstat_getfile2() call is recommended. This call returns information about open files for a specified process. For the specified process, there is one instance of this context for each open file descriptor. For each instance, data up to a maximum of elemsize bytes are returned in the structs pst_fileinfo pointed to by buf. The elemcount parameter specifies the number of structs pst_fileinfo that are available at buf. The index parameter specifies the starting index within the

context of open files for the specified process: it is a 32-bit quantity constructed of the pst_idx field of the 'owning' process, obtained via pstat_getproc(), as the most significant 16 bits, and the index of open files within the process as the least significant 16 bits. Example:
index = ((pst_idx << 16) | (file_index & 0xffff));

As a shortcut, information for a single file within the specified process may be obtained by setting elemcount to zero and setting the least significant 16 bits to the file descriptor number. The most significant 16 bits are still set to the pst_idx field from the pst_status structure for the process. The pst_fileinfo structure contains both psf_offset and psf_offset64 elements. The psf_offset element can correctly store only a 32-bit value, whereas the psf_offset64 element can store a 64-bit value. pstat_getfile() will fill in both psf_offset and psf_offset64 if the value can be correctly stored in both elements. If the offset is too large to be correctly stored in psf_offset, then psf_offset will contain a –1. No error will be set in this case.

Return Value:

This call returns –1 on failure or number of instances copied to buf on success.

Errors:

- ENOENT is set if the specified file is not found or closed.
- EINVAL is set if elemcount is not zero and index is less than zero.
- EINVAL is set if the user's size declaration is invalid.
- ESRCH is the specified process is not found.

4.6 pstat_getfile2()

Synopsis:

```
int pstat_getfile2(struct pst_fileinfo2 *buf, size_t elemsize, size_t elemcount, int index,
                   pid_t pid);
```

Description:

This call returns information about open files for a specified process. For the specified process, there is one instance of this context for each open file descriptor. For each instance, data up to a maximum of elemsize bytes are returned in the structs pst_fileinfo2 pointed to by buf. The elemcount parameter specifies the number of structs pst_fileinfo2 that are available at buf. The index parameter specifies the starting index within the context of open files for the specified process: It is the file descriptor number with which to begin. The pid parameter specifies the process ID (IDentification number).

As a shortcut, information for a single file within the specified process may be obtained by setting elemcount to zero and setting the index to the file descriptor number. The target PID should be specified in the pid argument. The pst_fileinfo2 structure contains both psf_offset and psf_offset64 element. The psf_offset element can correctly store only a 32-bit value, whereas the psf_offset64 element can store a 64-bit value. pstat_getfile2() will fill in both psf_offset and psf_offset64 if the value can be correctly stored in both elements. If the offset is too large to be correctly stored in psf_offset , then psf_offset will contain a -1. No error will be set in this case.

Return Value:

This call returns –1 on failure or number of instances copied to buf on success.

Errors:

- ENOENT is set if the specified file is not found or closed.
- EINVAL is set if elemcount is not zero and index is less than zero.
- EINVAL is set if the user's size declaration is invalid.
- ESRCH is the specified process is not found.

Example:
List all the open files for the parent process.

```
#include <sys/pstat.h>

void main(void)
{
#define BURST ((size_t)10)
      pid_t target = getppid();
      struct pst_fileinfo2 psf[BURST];
      int i, count;
      int idx = 0; /* index within the context */

      printf("Open files for process ID %d\n", target);

      /* loop until all fetched */
      while ((count = pstat_getfile2(psf, sizeof(struct pst_fileinfo2),
                                     BURST, idx, target)) > 0) {
            /* Process them (max of BURST) at a time */
            for (i = 0; i < count; i++) {
                  printf("fd #%d\tFSid %x:%x\tfileid %d\n",
                              psf[i].psf_fd,
                              psf[i].psf_id.psf_fsid.psfs_id,
                              psf[i].psf_id.psf_fsid.psfs_type,
                              psf[i].psf_id.psf_fileid);
            }

            /*
             * Now go back and do it again, using the
             * next index after the current 'burst'
             */
            idx = psf[count-1].psf_fd + 1;
      }
      if (count == -1)
            perror("pstat_getfile2()");
}
```

4.7 pstat_getfiledetails()

Synopsis:
int pstat_getfiledetails(struct pst_filedetails *buf, size_t elemsize, struct pst_fid *fid);
Description:

This call provides detailed information about a particular open file. For each call, data up to a maximum of elemsize bytes are returned in the struct pst_filedetails pointed to by buf. The fid parameter uniquely identifies the file. This fid is obtained from calls to pstat_getfile2(), pstat_getproc(), or pstat_getprocvm(). The pst_filedetails structure contains information equivalent to the stat(2) call. Use of this function is limited to UID == 0 or effective UID match. Effective UID match occurs when the effective or real UID of the calling thread matches the effective or real UID of the target process and the target process has not done a set[u/g]id.

The structure members psfd_mode, psfd_ino, psfd_dev, psfd_uid, psfd_gid, psfd_atime, psfd_mtime, psfd_ctime will have meaningful values for regular files, character or block special files, and pipes. The value of the member psfd_nlink will be set to number of links to the file. The member psfd_rdev will have meaningful value for character or block special files, while the psfd_size is valid for regular files. Psfd_hi_fileid, psfd_lo_fileid, psfd_hi_nodeid, and

psfd_lo_nodeid are unique ids representing the opened file. These ids together are used to match the corresponding ids returned from the pstat_getfile2(). This call does not work for sockets other than AF_UNIX family type.

Return Value:

This call returns –1 on failure or 1 on success.

Errors:

- EFAULT is set if buf or fid points to invalid address.
- ENOENT is set if the specified file is not found or closed.
- EINVAL is set if the user's size declaration is invalid.
- EINVAL is set if the fid corresponds to a socket other than AF_UNIX family.
- ESRCH is the specified process is not found.
- EACESS is set if there is no effective UID match.
- EOVERFLOW is set if a value to be stored overflows one of the members of the pst_filedetails structure. The psfd_valid member indicates the field that overflowed.

Example:

```
#include <sys/pstat.h>

void main(void)
{
      struct pst_fileinfo2 psf;
      struct pst_filedetails psfdetails;
      int count, fd;

      fd = open("/stand/vmunix", O_RDONLY);
      count = pstat_getfile2(&psf, sizeof(psf), 0, fd, getpid());
      if (count == 1) {
            count = pstat_getfiledetails(&psfdetails,
                                         sizeof(psfdetails),
                                         &psf.psf_fid);
            if (count == 1) {
                  if ((psfdetails.psfd_hi_fileid == psf.psf_hi_fileid)&&
                      (psfdetails.psfd_lo_fileid == psf.psf_lo_fileid)&&
                      (psfdetails.psfd_hi_nodeid == psf.psf_hi_nodeid)&&
                      (psfdetails.psfd_lo_nodeid == psf.psf_lo_nodeid)){
                        printf("Success\n");
                  } else {
                        printf("State changed\n");
                  }
            } else {
                  perror("pstat_getfiledetails()");
            }
      } else {
            perror("pstat_getfile2");
      }
      close(fd);
}
```

4.8 pstat_getipc()

Synopsis:

int pstat_getipc(struct pst_ipcinfo *buf, size_t elemsize, size_t elemcount, int index);

Description:

This call returns the system-wide global System V IPC constants. These are currently defined at boot time, but may become dynamic in future releases. There is only one instance of this context. Data up to a maximum of elemsize bytes are returned in the struct pst_ipcinfo pointed to by buf. The elemcount parameter must be 1. The index parameter must be 0.

Return Value:

This call returns –1 on failure or 1 on success.

Errors:

- EINVAL is set if elemcount is not equal to 1 or index is not equal to 0.
- EINVAL is set if user's size declaration is not valid.
- EFAULT is set if buf is invalid.
- EOVERFLOW is set if any of the fields in narrow mode exceed the maximum positive integer value. The corresponding bit in the valid vector is cleared.

Example:

```
#include <sys/pstat.h>

void main(void)
{
        struct pst_ipcinfo psi;

        if (pstat_getipc(&psi, sizeof (psi), 1, 0) == -1) {
                printf("\nError getting ipc info");
                return;
        }
        printf("\n Maximum value for semaphore: %ld", psi.psi_semvmx);
        printf("\n System wide total semaphores: %ld", psi.psi_semmns);
        printf("\n Maximum shared memory segment size %ld", psi.psi_shmmax);
}
```

4.9 pstat_getlv()

Synopsis:

int pstat_getlv(struct pst_lvinfo *buf, size_t elemsize, size_t elemcount, int index);

Description:

This call returns information about logical volume configured in the system. Each structure returned describes one logical volume. For each instance data up to a maximum of elemsize bytes are returned in the structs pst_lvinfo pointed to by buf. The elemcount parameter specifies the number of structs pst_lvinfo that are available at buf. The index parameter specifies the starting index within the context of logical volumes. As a shortcut, information for a single logical volume may be obtained by setting elemcount to zero and setting index to the dev_t of that logical volume.

Return Value:

This call returns –1 on failure or number of instances copied to buf on success.

Errors:

- If the user's size declaration isn't valid, EINVAL is set.
- EINVAL is set if the initial selection, index, isn't valid.
- ESRCH is set if the specific-LV shortcut is used and there is no logical volume with that device specification,
- EFAULT is set if buf points to invalid address.

Example:

```
#include <stdlib.h>
#include <errno.h>
#include <sys/pstat.h>

void main(void)
{
      int i;
      int rv;
      struct pst_dynamic psd;
      struct pst_lvinfo *pstlv;
      size_t NumOpenLV = 20;

      pstlv = (struct pst_lvinfo *)
              malloc(NumOpenLV * sizeof(struct pst_lvinfo));
      if (pstlv == (struct pst_lvinfo *) 0) {
            fprintf(stderr, "iomon: memory allocation failure\n");
            exit(1);
      }
      if ((rv = pstat_getlv(pstlv, sizeof(struct pst_lvinfo), NumOpenLV, 0))
            == -1) {
            printf("pstat_getlv() returned -1\n");
            printf("Error code = %d\n", errno);
            exit(1);
      }
      for (i = 0; i < rv;  i++) {
            printf("%3d --> Major = %d; Minor = 0x%06x; Reads = %lu\n",
                        i, (int) pstlv[i].psl_dev.psd_major,
                        (int) pstlv[i].psl_dev.psd_minor,
                        (int) pstlv[i].psl_rxfer);
      }
}
```

4.10 pstat_getlwp()

Synopsis:

int pstat_getlwp(struct lwp_status *buf, size_t elemsize, size_t elemcount, int index, pid_t pid);

Description:

This call returns information about the threads or LWPs (Lightweight Processes) in a process. There is one instance of this context for each LWP in a process on the system. For each instance requested, data up to a maximum of elemsize bytes are returned in the struct lwp_status pointed to by buf. The elemcount parameter specifies the number of structs lwp_status that are available at buf. The index parameter specifies the starting index within the context of LWPs in a process.

If pid is set to -1 and elemcount is greater than 0, elemcount entries of system LWP information are returned to the caller program. If pid is greater than or equal to 0 and elemcount is greater than 0, elemcount entries of LWP info within the process specified by pid are returned. As a shortcut, information about a single LWP can be obtained by setting elemcount to zero and setting index to the TID (Thread ID) of that LWP within its process.

Return Value:

This call returns –1 on failure or number of instances copied to buf on success.

Errors:

- EINVAL is set if the following conditions are not met.

(1) elemsize is > 0 or <= actual size of the kernel's version
(2) index >= 0
(3) if pid >= 0, elemcount >= 0
(4) if pid == -1, elemcount > 0

- ESRCH is set if the specific-PID (Process Identification number) shortcut is used and there is no active process with that PID.
- EFAULT is set of buf points to invalid address.

Example:

```
#include <sys/pstat.h>

void main(void)
{
      struct lwp_status lwpbuf;

      /*
       * Get information for LWP whose lwpid is 4321 within
       * a process whose pid is 1234.
       */
      count = pstat_getlwp(buf, sizeof(struct lwp_status), 0, 4321,
                           1234);
      if (count == -1) {
            perror("pstat_getlwp()");
      } else {
            . . . . . .
      }
}
```

4.11 pstat_getmpathname()

Synopsis:

```
int pstat_getmpathname(struct pst_mpathnode *buf, size_t elemsize, size_t elemcount,
                       int index, struct psfsid *fsid);
```

Description:

This call returns entries from the system cache of recent directory and file names looked up (DNLC) for a specified file system. The fsid parameter uniquely identifies the file system. This fsid should be the psf_fsid field of a psfileid structure obtained from calls to pstat_getfile2(), pstat_getproc(), or pstat_getprocvm(). The index parameter specifies the starting entry within the chain of DNLC entries to be returned for the specified file system. The elemcount parameter specifies the number of DNLC entries to be returned. Typically, the index parameter will be specified as zero and the elemcount parameter will be equal to the dnlc size obtained through pstat_getdynamic() call. For each call, data up to a maximum of elemsize bytes are returned in the structs pst_mpathnode pointed to by buf.

Reverse pathname lookup can be performed by searching the entries for one that has a psr_file member equal to the psr_parent member of the current entry. This is done until an entry with a NULL psr_parent entry is located, which indicates that the entry for the root of the file system has been found. The pathname from the root of the file system is formed by prefixing the names given by the psr_name member of each of the entries found during the process. If desired, the full pathname can then be formed by concatenating the pathname to the mount point of the file system.

Use of this function is limited to UID == 0.

Return Value:

On success, the function returns the number of DNLC entries copied. In case of failure, the value of -1 is returned and errno is set indicating the cause of the failure.

Errors:

- EPERM is set if UID != 0.
- EFAULT is set if buf or fsid point to invalid address.
- EINVAL is set if elemcount is not greater than or equal to 1 or index is not greater than or equal to 0.
- EINVAL is set if the user's size declaration isn't valid.
- ENOENT is set if the specified file system is not found or does not have DNLC entries.

Example:

```
#include <sys/pstat.h>

void main(void)
{
      struct pst_fileinfo2 psf;
      struct pst_mpathnode mpath_buf[20];
      int i, rv, count, fd;
      pid_t target;

      target = getpid();
      fd = open("/etc/passwd", O_RDONLY);
      rv = pstat_getfile2(&psf, sizeof(psf), 0, fd, target);

      if (rv == 1) {

            /*
             * Ask for multiple pathname information.
             */
            count = pstat_getmpathname(mpath_buf,
                                       sizeof(struct pst_mpathnode),
                                       20, 0, &(psf.psf_id.psf_fsid));
            if (count > 0) {
                  for (i = 0; i < count; i++) {
                        printf("component %d: %s\n",
                                    i, mpath_buf[i].psr_name);
                  }
            } else if (count == 0) {
                  printf("pathnames not found in system cache\n");
            } else {
                  perror("pstat_getmpathname()");
            }
      } else {
            perror("pstat_getfile2");
      }
      close(fd);
}
```

4.12 pstat_getmsg()

Synopsis:

int pstat_getmsg(struct pst_msginfo *buf, size_t elemsize, size_t elemcount, int index);

Description:

This call returns information describing System V message queues. Each structure returned describes one message queue identifier on the system. For each instance data up to a maximum of elemsize bytes are returned in the structs pst_msginfo pointed to by buf. The elemcount parameter specifies the number of structs pst_msginfo that are available at buf. The index parameter specifies the starting index within the context of System V message queues. As a shortcut, information for a single message queue may be obtained by setting elemcount to zero and setting index to the msqid of that message queue.

Return Value:

This call returns –1 on failure or number of instances copied to buf on success.

Errors:

- EINVAL is set if the user's size declaration isn't valid.
- EINVAL is set if the initial selection, index, isn't valid.
- ESRCH is set if the specific-msqid shortcut is used and there is no message queue with that ID
- EACCES is set if the specific-msqid shortcut is used and the caller does not have read permission to the message queue.
- EFAULT is set if buf points to invalid address.

Example:

```
#include <sys/pstat.h>
#include <stdlib.h>
#include <errno.h>

void main(void)
{
      struct pst_ipcinfo psi;
      struct pst_msginfo *psmp;
      long msgmni;
      int qs_inuse;

      /* Get total message Qs */
      if ((pstat_getipc(&psi, sizeof(psi), 1, 0) == -1) &&
          (errno != EOVERFLOW)) {
            printf("\nCould not get ipc info");
            exit(1);
      }
      msgmni = psi.psi_msgmni;
      psmp = (struct pst_msginfo *)
             malloc(msgmni * sizeof(struct pst_msginfo));

      /* Get Qs in use */
      qs_inuse = pstat_getmsg(psmp, sizeof(struct pst_msginfo), msgmni, 0);
      if (qs_inuse == -1) {
            printf("\n Could not get msg info");
            exit(1);
      }
      free(psmp);
      printf("\n Total Qs = %d", msgmni);
      printf("\n Qs in use = %d", qs_inuse);
      printf("\nQs not in use = %d\n", msgmni - qs_inuse);
}
```

4.13 pstat_getnode()

Synopsis:

int pstat_getnode(struct pst_node *buf, size_t elemsize, size_t elemcount, int index);
Description:

This call returns information about SCA (ccNUMA) system nodes. It is only available as 64-bit data (i.e., _PSTAT64 needs to be defined). There is one instance of this context for each SCA node on the system. For each instance data up to a maximum of elemsize bytes are returned in the struct pst_node pointed to by buf. The elemcount parameter specifies the number of struct pst_node that are available at buf. The index parameter specifies the starting logical node ID that is requested.

Return Value:

This call returns –1 on failure or number of instances copied to buf on success.

Errors:

- EINVAL is set if the user's size declaration isn't valid.
- EINVAL is set if the initial selection, index, isn't valid.
- EFAULT is set if buf points to invalid address.

4.14 pstat_getpathname()

Synopsis:

int pstat_getpathname(char *buf, size_t elemcount, struct pst_fid *fid);

Description:

This call provides the full pathname of an opened file in buf if it is available in the system cache of recent names looked up (DNLC). The fid parameter uniquely identifies the opened file. This fid is obtained from calls to pstat_getfile(), pstat_getfile2(), pstat_getproc(), or pstat_getprocvm(). The value of elemcount should be at least one greater than the length of the pathname to be returned. The PATH_MAX variable from pathconf(2) can be used for this purpose. Use of this function is limited to UID == 0 or effective UID match. Please refer pstat_getfiledetails() call for more information on Effective UID match. This call does not work for sockets.

Return Value:

On success, the function returns the length of the pathname copied starting at the location specified by buf. If the pathname is not available in the system cache, 0 is returned and errno is not set. On other failures, the value of -1 is returned and errno is set indicating the cause of the failure.

Errors:

- EFAULT is set if buf/fid points to an invalid address.
- ENOENT is set if the file is not found, or it is closed.
- EINVAL is set if elemcount is less than or equal to zero
- EINVAL is set if called for a socket.
- EACCESS is set if effective ID does not match.
- EOVERFLOW is set if the elemcount parameter is not at lease one greater than the length of the pathname to be returned.

Example:

```
#include <sys/pstat.h>

void main(void)
{
      struct pst_fileinfo2 psf;
      char filename[20];
```

```
        int rv, count, fd;

        fd = open("/etc/passwd", O_RDONLY);
        rv = pstat_getfile2(&psf, sizeof(psf), 0, fd, getpid());

        if (rv == 1) {

                /*
                 * Ask for pathname information.
                 */
                count = pstat_getpathname(filename, 20, &(psf.psf_fid));
                if (count > 0) {
                        if (strncmp("/etc/passwd", filename, count) == 0) {
                                printf("Success\n");
                        } else {
                                printf("Error encountered\n");
                        }
                } else if (count == 0) {
                        printf("pathname not found in system cache\n");
                } else {
                        perror("pstat_getpathname()");
                }
        } else {
                perror("pstat_getfile2");
        }
        close(fd);
}
```

4.15 pstat_getproc()

Synopsis:
int pstat_getproc(struct pst_status *buf, size_t elemsize, size_t elemcount, int index);
Description:

This call returns information about active processes in the system. There is one instance of this context for each active process on the system. For each instance, data up to a maximum of elemsize bytes are returned in the structs pst_status pointed to by buf. The elemcount parameter specifies the number of structs pst_status that are available at buf. The index parameter specifies the starting index within the context of processes. As a shortcut, information for a single process may be obtained by setting elemcount to zero and setting index to the PID of that process.

Return Value:

This call returns –1 on failure or number of instances copied to buf on success.

Errors:

- EINVAL is set if the user's size declaration isn't valid.
- EINVAL is set if the initial selection, offset, isn't valid.
- ESRCH is set if the specific-PID shortcut is used and there is no active process with that PID.
- EFAULT is set if buf points to an invalid address.

Examples:

```
#include <sys/pstat.h>

/* Get information about all processes -- 10 at a time. */
void main(void)
{
```

```
#define BURST ((size_t)10)
      struct pst_status pst[BURST];
      int i, count;
      int idx = 0; /* index within the context */

     /* loop until count == 0 */
      while ((count = pstat_getproc(pst, sizeof(pst[0]), BURST, idx))
                  > 0) {
            /* got count (max of BURST) this time.  Process them */
            for (i = 0; i < count; i++) {
                  printf("pid is %d, command is %s\n", pst[i].pst_pid,
                          pst[i].pst_ucomm);
            }
            /*
             * Now go back and do it again, using the next index after
             * the current 'burst'
             */
            idx = pst[count-1].pst_idx + 1;
      }

      if (count == -1) {
            perror("pstat_getproc()");
      }
}

/* Get a particular process' information. */
void main(void)
{
      struct pst_status pst;
      int target = (int)getppid();

      if (pstat_getproc(&pst, sizeof(pst), (size_t)0, target) != -1) {
            printf("Parent started at %s", ctime(&pst.pst_start));
      } else {
            perror("pstat_getproc");
      }
}
```

4.16 pstat_getprocessor()

Synopsis:
int pstat_getprocessor(struct pst_processor *buf, size_t elemsize, size_t elemcount, int index);
Description:

This call returns information about processors in the system. Each structure returned describes one processor on a multi-processor system. A total of one structure is returned for uni-processor machines. For each instance data up to a maximum of elemsize bytes are returned in the structs pst_processor pointed to by buf. The elemcount parameter specifies the number of structs pst_processor that are available at buf. The index parameter specifies the starting index within the context of processors.

Return Value:

This call returns –1 on failure or number of instances copied to buf on success.

Errors:

- EINVAL is set if the user's size declaration isn't valid.
- EINVAL is set if index < 0.

- EFAULT is set if buf points to an invalid address.

Example:

Get information about all processors, first obtaining number of processor context instances.

```
#include <sys/pstat.h>
#include <stdlib.h>

void main(void)
{
      struct pst_dynamic psd;
      struct pst_processor *psp;
      int count;

      if (pstat_getdynamic(&psd, sizeof(psd), (size_t)1, 0) != -1) {
            size_t nspu = psd.psd_proc_cnt;
            psp = (struct pst_processor *)
                   malloc(nspu * sizeof(struct pst_processor));
            count = pstat_getprocessor(psp, sizeof(struct pst_processor),
                                       nspu, 0);
            if (count > 0) {
                  int i;
                  int total_execs = 0;

                  for (i = 0; i < count; i++) {
                        int execs = psp[i].psp_sysexec;
                        total_execs += execs;
                        printf("%d exec()s on processor #%d\n", execs,i);
                  }

                printf("total execs for the system were %d\n",
                         total_execs);
            }else {
                  perror("pstat_getprocessor");
            }
      } else {
            perror("pstat_getdynamic");
      }
}
```

4.17 pstat_getprocvm()

Synopsis:

int pstat_getprocvm(struct pst_vm_status *buf, size_t elemsize, size_t elemcount, int index);

Description:

This call provides information about process' address space. At most one instance (process region) is returned for each call to pstat_getprocvm(). For each instance, data up to a maximum of elemsize bytes are returned in the struct pst_vm_status pointed to by buf. The elemcount parameter identifies the process for which address space information is to be returned. An elemcount parameter of zero indicates that address space information for the currently executing process should be returned. Information for a specific process (other than currently executing one) may be obtained by setting elemcount to the PID of that process. The index parameter specifies the relative index (beginning with 0) within the context of process regions for the indicated process. For example, an index of 3 indicates the 4th process region within the

indicated process' address space. Additional information on VM regions mapped to files can be obtained with the pstat_getfiledetails() call.

Return Value:

This call returns 1 on success or –1 on failure. It returns 0 when there are no regions to copy.

Errors:

- EINVAL is set if the user's size declaration isn't valid.
- EINVAL is set if index is less than 0.
- ESRCH is set if elemcount > 0, and there is no active process with that PID.
- EFAULT is set if buf points to an invalid address.

Example:

```
#include <sys/pstat.h>

void main(void)
{
      struct pst_vm_status pst;
      int count1, count2, i = 0;
      char name[100];

      /* get info on all memory regions */
      while (1) {
            count1 = pstat_getprocvm(&pst, sizeof(pst), getpid(), i);
            if (count1 == 0)
                  break;
            if (count1 == -1){
                  perror("\n pstat_getprocvm");
                  break;
            }

            printf("\n type: %d, length: %d, phys pages: %d, ref count: %d",
                        pst.pst_type, pst.pst_length,
                        pst.pst_phys_pages, pst.pst_refcnt);
            count2 = pstat_getpathname(name, 100, &pst.pst_fid);
            if (count2 > 0) {
                  printf(" %s", name);
            }
            i++;
      }
      printf("\n");
}
```

4.18 pstat_getsem()

Synopsis:

int pstat_getsem(struct pst_seminfo *buf, size_t elemsize, size_t elemcount, int index);

Description:

This call provides information about System V semaphore sets. Each structure returned describes one semaphore set on the system. For each instance data up to a maximum of elemsize bytes are returned in the structs pst_seminfo pointed to by buf. The elemcount parameter specifies the number of structs pst_seminfo that are available at buf. The index parameter specifies the starting index within the context of System V semaphore sets. As a shortcut,

information for a single semaphore set may be obtained by setting elemcount to zero and setting index to the semid of that semaphore set.

Return Value:

This call returns –1 on failure or number of instances copied to buf on success.

Errors:

- EINVAL is set if the user's size declaration isn't valid.
- EINVAL is set if index < 0.
- ESRCH is set if the specific-semid shortcut is used and there is no semaphore set with that ID.
- EFAULT is set if buf points to invalid address.

Example:

```
#include <sys/pstat.h>
#include <stdlib.h>

void main(void)
{
      struct pst_ipcinfo psi;
      struct pst_seminfo *pssp;
      size_t semmni;
      int i, semsets_inuse, sems_inuse = 0;

      if ((pstat_getipc (&psi, sizeof(psi), 1, 0) == -1) &&
          (errno != EOVERFLOW)) {
            printf ("\n Could not get ipc info\n");
            exit(1);
      }
      semmni = psi.psi_semmni;
      pssp = (struct pst_seminfo *)
             calloc(semmni, sizeof (struct pst_seminfo));
      if (!pssp) {
            printf("\n No memory\n");
            exit(1);
      }
      semsets_inuse = pstat_getsem(pssp,
                                   sizeof(struct pst_seminfo), semmni, 0);
      for (i = 0; i < semsets_inuse; i++) {
            sems_inuse += pssp[i].pse_nsems;
      }
      free (pssp);
      printf("\n Total Semaphores sets: %d", semmni);
      printf("\n Active Semaphores sets: %d", semsets_inuse);
      printf("\n Total Semaphores: %d", psi.psi_semmns);
      printf("\n Active Semaphores: %d", sems_inuse);
}
```

4.19 pstat_getshm()

Synopsis:

int pstat_getshm(struct pst_shminfo *buf, size_t elemsize, size_t elemcount, int index);

Description:

Returns information about System V shared memory segments. Each structure returned describes one segment identifier on the system. For each instance data up to a maximum of elemsize bytes are returned in the structs pst_shminfo pointed to by buf. The elemcount parameter specifies the number of structs pst_shminfo that available at buf. The index parameter

specifies the starting index within the context of System V shared memory segments. As a shortcut, information for a single shared memory segment may be obtained by setting elemcount to zero and setting index to the shmid of that shared memory segment.

Return Value:

This call returns –1 on failure or number of instances copied to buf on success.

Errors:

- EINVAL is set if the user's size declaration isn't valid.
- EINVAL is set if index < 0.
- ESRCH is set if the specific-shmid shortcut is used and there is no shared memory segment with that ID.
- EFAULT is set if buf points to invalid address.

Example:

```
Get information about all shared memory segments.
#include <sys/pstat.h>
#include <stdlib.h>

void main(void)
{
      struct pst_ipcinfo psi;
      struct pst_shminfo *pss;

      if (pstat_getipc(&psi, sizeof(psi), (size_t)1, 0) != -1) {
            size_t num_shm = psi.psi_shmmni;
            pss = (struct pst_shminfo *)
                   malloc(num_shm * sizeof(struct pst_shminfo));
            if (!pss) {
                  printf("\n No memory");
                  exit(1);
            }
            if (pstat_getshm(pss, sizeof(struct pst_shminfo), num_shm, 0)
                != -1) {
                  int i;
                  printf("owner\tkey\tsize\n");
                  for (i = 0; i < num_shm; i++) {
                        /* skip inactive segments */
                        if (!(pss[i].psh_flags & PS_SHM_ALLOC))
                              continue;
                        printf("%l\t%#x\t%d\n",
                              (long) pss[i].psh_uid, pss[i].psh_key,
                              pss[i].psh_segsz);
                  }
            } else {
                  perror("pstat_getshm");
            }
            free(pss);
      } else {
            perror("pstat_getipc");
      }
}
```

4.20 pstat_getsocket()

Synopsis:

int pstat_getsocket(struct pst_socket *buf, size_t elemsize, struct pst_fid *fid);

Description:

This call returns detailed information specific to a socket. For the specified socket, there is one instance of this context. For each call, data up to a maximum of elemsize bytes are returned in the struct pst_socket pointed to by buf. The fid parameter uniquely identifies the socket. This fid is obtained from calls to pstat_getfile2(). Use of this call is limited to UID == 0 or effective UID match. Please refer to pstat_getfiledetails() call for more information on effective UID match.

For AF_UNIX sockets that are opened to files, more information about those files can be obtained with the pstat_getfiledetails() call. In case of AF_UNIX sockets, pst_peer_hi_nodeid and pst_peer_lo_nodeid fields can be used to find the peer socket by matching them with pst_hi_nodeid and pst_lo_nodeid. The members pst_boundaddr and pst_remaddr contain data of the form struct sockaddr, sockaddr_un, sockaddr_in, or sockaddr_in6 depending on the socket family. These need to be copied to the structure they represent using pst_boundaddr_len and pst_remaddr_len to assure proper structure alignment.

Return value:

On success, the call returns 1. On failure, value of -1 is returned and errno is set indicating the cause of the failure.

Errors:

- EINVAL is set if the user's size declaration isn't valid.
- EACESS is set if no effective ID match.
- EFAULT is set if buf points to an invalid address.
- EINVAL is set if the file is not a socket.
- ENOENT is set if the specified socket is not found or is being closed.
- ESRCH is set if the required process is not found.

Example:

```
#include <sys/pstat.h>
#include <socket.h>

void main(void)
{
      struct pst_fileinfo2 psf;
      struct pst_socket psfsocket;
      int rv, count, fd;

      fd = socket(AF_INET, SOCK_STREAM, 0);
      rv = pstat_getfile2(&psf, sizeof(psf), 0, fd, getpid());

      if ((rv == 1) && (psf.psf_type == PS_TYPE_SOCKET)){
            count = pstat_getsocket(psfsocket, sizeof(struct pst_socket),
                                    &(psf.psf_fid));
            if (count == 1) {
                  if ((psfsocket.pst_hi_fileid == psf.psf_hi_fileid) &&
                      (psfsocket.pst_lo_fileid == psf.psf_lo_fileid) &&
                      (psfsocket.pst_hi_nodeid == psf.psf_hi_nodeid) &&
                      (psfsocket.pst_lo_nodeid == psf.psf_lo_nodeid)) {
                        printf("The type of socket is %d, should be %d\n",
                                    psfsocket.pst_type, PS_SOCK_STREAM);
                  } else {
                        printf("State changed\n");
                  }
            } else {
```

```
                perror("pstat_getsocket()");
        }
    } else {
        perror("pstat_getfile2");
    }
    close(fd);
}
```

4.21 pstat_getstable()

Synopsis:

int pstat_getstable(struct pst_stable *buf, size_t elemsize, size_t elemcount, int index);

Description:

Returns information contained in the system's stable storage area. There is one instance of this context. Data up to a maximum of elemsize bytes are returned in the struct pst_stable pointed to by buf. The elemcount parameter must be 1. The index parameter must be 0.

Return Value:

On success, the call returns 1. On failure, value of -1 is returned and errno is set indicating the cause of the failure.

Errors:

- EINVAL is set if elemcount is not 1 or index is not 0.
- EINVAL is set if the user's size declaration isn't valid.
- ENOSYS is set if stable storage is not supported.
- EFAULT is set if buf points to an invalid address.

4.22 pstat_getstatic()

Synopsis:

int pstat_getstatic(struct pst_static *buf, size_t elemsize, size_t elemcount, int index);

Description:

This call returns static system information -- data that will remain the same (at least) until reboot. It also provides the current sizes of all pstat data structures. There is one instance of this context. Data up to a maximum of elemsize bytes are returned in the struct pst_static pointed to by buf. The elemcount parameter must be 1. The index parameter must be 0.

Return Value:

On success, the call returns 1. On failure, value of -1 is returned and errno is set indicating the cause of the failure.

Errors:

- EINVAL is set if elemcount is not 1 or index is not 0.
- If the user's size declaration isn't valid, EINVAL is set.
- If buf points to an invalid address, EFAULT is set.

Example:

```
#include <sys/pstat.h>

void main(void)
{
    struct pst_static pst;

    if (pstat_getstatic(&pst, sizeof(pst), (size_t)1, 0) != -1) {
        printf("page size is %d bytes\n", pst.page_size);
```

```
	} else {
		perror("pstat_getstatic");
	}
}
```

4.23 pstat_getstream()

Synopsis:

int pstat_getstream(struct pst_stream *buf, size_t elemsize, size_t elemcount, int moduleskip,
struct pst_fid *fid);

Description:

This call provides detailed information specific to a stream. For the specified stream, there is one instance of this context for the stream head, each module, and the driver. For each call, data up to a maximum of elemsize bytes are returned in the structs pst_stream pointed to by buf. The elemcount parameter specifies the number of structs pst_stream that are available at buf. The moduleskip parameter indicates the number of modules to skip before returning information about any modules. Head information is returned for every call. The fid parameter uniquely identifies the file. This is obtained from calls to pstat_getfile2(). Use of this function is limited to UID == 0 or effective UID match. Please refer pstat_getfiledetails() call for more information on Effective UID match.

Return Value:

On success, the function returns the number of structures copied. This is at least 1 as the head information is always returned. On failure, -1 is returned and errno is set indicating the cause of the failure.

Errors:

- EINVAL is set If the user's size declaration isn't valid,.
- EINVAL is set if num parameter is <= 0 or moduleskip is < 0.
- EACESS is set f no effective ID match.
- EFAULT is set if buf points to an invalid address.
- ENOSTR is set if the file is neither stream type nor a stream-based socket.
- ENOENT is returned if file not found.
- ESRCH is returned if the required process is not found.

Example:

```
#include <sys/pstat.h>
#include <fcntl.h>

void main(void)
{
	struct pst_fileinfo2 psf;
	struct pst_stream psfstream[3];
	int rv, count, fd;

	fd = open("/dev/echo", O_RDONLY);
	rv = pstat_getfile2(&psf, sizeof(psf), 0, fd, getpid());

	if ((rv == 1) && (psf.psf_type == PS_TYPE_STREAMS)){

	/*
	 * Ask for 3 structures (head + module(s) + driver).
	 * If there are no modules, we expect 2 structures(head, driver)
	 * If there is 1 module, we expect 3 structures (head, module, driver)
```

```
        * If there is more than 1 module, we expect 3 structures,
        * head, modules).
        */
             count = pstat_getstream(psfstream, sizeof(struct pst_stream),
                              sizeof(psfstream) / sizeof(struct pst_stream),
                              0, &psf.psf_fid);
             if (count > 0) {
                   if ((psfstream[0].val.head.pst_hi_fileid ==
                                                    psf.psf_hi_fileid) &&
                         (psfstream[0].val.head.pst_lo_fileid ==
                                                    psf.psf_lo_fileid) &&
                         (psfstream[0].val.head.pst_hi_nodeid ==
                                                    psf.psf_hi_nodeid) &&
                         (psfstream[0].val.head.pst_lo_nodeid ==
                                                    psf.psf_lo_nodeid)) {
                               printf("Major number of the stream is %d\n",
                                       psfstream[0].val.head.pst_dev_major);
                   } else {
                         printf("State changed\n");
                   }
             } else {
                   perror("pstat_getstream()");
             }
       } else {
             perror("pstat_getfile2");
       }
       close(fd);
}
```

4.24 pstat_getswap()

Synopsis:
int pstat_getswap(struct pst_swapinfo *buf, size_t elemsize, size_t elemcount, int index);
Description:

This call returns swap area information. There is one instance of this context for each swap area (block or file system) configured into the system. For each instance data up to a maximum of elemsize bytes are returned in the structs pst_swapinfo pointed to by buf. The elemcount parameter specifies the number of structs pst_swapinfo that are available at buf. The index parameter specifies the starting index within the context of swap areas.

Return Value:

On success, the call returns 1. On failure, -1 is returned and errno is set indicating the cause of the failure.

Errors:

- EINVAL is set if elemcount == 0 or index < 0.
- EINVAL is set if the user's size declaration isn't valid.
- EFAULT is set if buf points to an invalid address.

4.25 pstat_getvminfo()

Synopsis:
int pstat_getvminfo(struct pst_vminfo *buf, size_t elemsize, size_t elemcount, int index);
Description:

This call returns information about the virtual memory. There is only one instance of this context. Data up to a maximum of elemsize bytes are returned in the struct pst_vminfo pointed to by buf. The elemcount parameter must be 1. The index parameter must be 0.

Return Value:

On success, the call returns 1. On failure, value of -1 is returned and errno is set indicating the cause of the failure.

Errors:

- EINVAL is set if elemcount is not 1 or index is not 0.
- EINVAL is set if the user's size declaration isn't valid.
- EFAULT is set if buf points to an invalid address.

Example:

```
#include <sys/pstat.h>

void main(void)
{
      struct pst_vminfo pst;
      int count;

      count = pstat_getvminfo(&pst, sizeof(pst), 1, 0);

      if (count == 1) {
            printf("\n Rate of faults: %d\n", pst.psv_rfaults);
      } else {
            perror("pstat_getvminfo()");
      }
}
```

5. Binary Compatibility

This section describes pstat wrappers' commitment to software compatibility, in terms of currently supported and future versions of HP-UX. The specific calling convention of passing the expected data structure size is used in order to allow for future expansion of the interface, while preserving source and binary compatibility for programs written using the pstat interfaces. Three rules are followed to allow existing applications to continue to execute from release to release of the operating system.

1. New data for a context are added to the end of that context's data structure.
2. Old, obsolete data members are not deleted from the data structure.
3. The operating system honors the *elemsize* parameter of the call and only returns the first *elemsize* bytes of the context data, even if the actual data structure has since been enlarged.

In this way, an application which passes its compile-time size of the context's data structure (for example, *sizeof(struct pst_processor)* for the per-processor context) as the elemsize parameter will continue to execute on future operating system releases without recompilation, even those that have larger context data structures. If the program is recompiled, it will continue to execute on that and future releases. Note that the reverse is not true: a program using the pstat interfaces compiled on, say, HP-UX release 10.0 will not work on HP-UX release 9.0.

6. Appendix

pst_crashdev

This structure describes a crash dump device. It is only available as 64-bit data. Header file: <sys/pstat/crash_pstat_body.h>

```
int64_t      psc_idx;           /* Index of this device. */
struct __psdev psc_device;      /* Device number of physical device */
int64_t      psc_offset;        /* Offset (kB) of area on physical device */
int64_t      psc_size;          /* Size (kB) of dump area */
struct psdev psc_lv;            /* Logical volume dev number, if any */
int64_t      psc_source;        /* How'd this device get configured? */
```

The psc_idx member+1 is used as offset for further pstat_getcrashdev() calls. Flag values for psc_source are:

PS_BOOTTIME → Device configured at boot.

PS_RUNTIME → Device configured by crashconf(2).

pst_crashinfo

This structure describes the system crash dump configuration. It is only available as 64-bit data. It contains the following members. Header file: <sys/pstat/crash_pstat_body.h>

```
int64_t      psc_flags;         /* Dump configuration flags */
struct __psdev psc_headerdev;   /* Device containing dump header */
int64_t      psc_headeroffset;  /* Byte Offset of dump header on device */
int64_t      psc_ncrashdevs;    /* Number of dump devices */
int64_t      psc_totalsize;     /* Total amount of dump space (kB) */
int64_t      psc_included;      /* Page classes to be included */
int64_t      psc_excluded;      /* Page classes to be excluded */
int64_t      psc_default;       /* Defaults for unspecified classes */
int64_t      psc_nclasses;      /* Number of classes */
int64_t      psc_pgcount[PST_MAXCLASSES];
                                /* Number of pages in each class */
```

The psc_flags member can take the following values:

PS_EARLY_DUMP → An early dump was taken

PS_CONF_CHANGED → Configuration changed since boot

PS_HEADER_VALID → headerdev and headeroffset are valid

pst_diskinfo

Header file: <sys/pstat/disk_pstat_body.h>

```
_T_LONG_T   psd_idx;        /* Index for further pstat() requests */
struct __psdev  psd_dev;    /* Device specification for the disk. Describes
                             * the block dev. Refer pstat_body.h header. */
_T_LONG_T   psd_dktime;     /* cumulative ticks on the disk */
_T_LONG_T   psd_dkseek;     /* cumulative number of seeks done */
_T_LONG_T   psd_dkxfer;     /* cumulative number of transfers; includes
                             * requests with high (read/write) and low
                             * (ioctl) priorities */
_T_LONG_T   psd_dkwds;      /* cumulative number of 64-byte transfers */
float       psd_dkmspw;     /* OBSOLETE: do not use */
struct __psdev  psd_cdev;   /* device specification for the raw disk */
```

```
struct psdrvnam psd_drv_name;   /* driver name */
_T_LONG_T   psd_token;      /* driver's ID */
_T_LONG_T   psd_instance;   /* the instance of the device */
struct __pshwpath psd_hw_path;  /* hardware path */
struct __psttime psd_dkwait; /* cumulative time from enqueue to start */
struct __psttime psd_dkresp; /* cumulative time from enqueue to done */
_T_LONG_T   psd_dkcyl_index; /* cylinder number index, used by sadp */
_T_LONG_T   psd_dkcyl[PS_DK_CYL_SIZE]; /* cylinder number array,
                                        * used by sadp */
_T_LONG_T   psd_dkqlen_curr;/* current queue length */
_T_LONG_T   psd_dkqlen;     /* cumulative queue length */
_T_LONG_T   psd_dkq_merged; /* cumulative # of transfer would have been
                             * if some of the requests weren't merged */
_T_LONG_T   psd_dkenq_cnt;  /* number of calls to enqueue */
_T_LONG_T   psd_status;     /* 0 = device is closed, 1 = device is open */
```

PS_DK_CYL_SIZE → 80

pst_dynamic

This structure contains dynamic system variables, ones which may change frequently during normal operation of the kernel. Heder file: <sys/pstat/global_pstat_body.h>

```
_T_LONG_T       psd_proc_cnt;     /* MP: number of active processors */
_T_LONG_T       psd_max_proc_cnt; /* MP: max active processors */
_T_LONG_T       psd_last_pid;   /* last run process ID */
_T_LONG_T       psd_rq;         /* run queue length */
_T_LONG_T       psd_dw;         /* jobs in disk wait */
_T_LONG_T       psd_pw;         /* jobs in page wait */
_T_LONG_T       psd_sl;         /* jobs sleeping in core */
_T_LONG_T       psd_sw;         /* swapped out runnable jobs */
_T_LONG_T       psd_vm;         /* total virtual memory */
_T_LONG_T       psd_avm;        /* active virtual memory */
_T_LONG_T       psd_rm;         /* total real memory */
_T_LONG_T       psd_arm;        /* active real memory */
_T_LONG_T       psd_vmtxt;      /* virtual memory text */
_T_LONG_T       psd_avmtxt;     /* active virtual memory text */
_T_LONG_T       psd_rmtxt;      /* real memory text */
_T_LONG_T       psd_armtxt;     /* active real memory text */
_T_LONG_T       psd_free;       /* free memory pages */
double          psd_avg_1_min;  /* global run queue lengths */
double          psd_avg_5_min;
double          psd_avg_15_min;
_T_LONG_T psd_cpu_time[PST_MAX_CPUSTATES];       /* OBSOLETE do not use */
double          psd_mp_avg_1_min[PST_MAX_PROCS]; /* OBSOLETE do not use */
double          psd_mp_avg_5_min[PST_MAX_PROCS]; /* OBSOLETE do not use */
double          psd_mp_avg_15_min[PST_MAX_PROCS];/* OBSOLETE do not use */
_T_LONG_T   psd_mp_cpu_time[PST_MAX_PROCS][PST_MAX_CPUSTATES]; /* OBSOLETE */
_T_LONG_T       psd_openlv;      /* # of open Logical Volumes */
_T_LONG_T       psd_openvg;      /* # of open LV Volume groups */
_T_LONG_T       psd_allocpbuf;   /* # of allocated LV pvol buffers */
_T_LONG_T       psd_usedpbuf;    /* # of LV pvol buffers in used */
_T_LONG_T       psd_maxpbuf;     /* max # of LV pvol buffers avail. */
_T_LONG_T       psd_activeprocs; /* # of active proc  table entries */
_T_LONG_T       psd_activeinodes;/* # of active inode table entries */
_T_LONG_T       psd_activefiles; /* # of active file  table entries */
```

```
_T_LONG_T       psd_mpdcnt;      /* # of (bad) memory pages deallocated */
_T_LONG_T       psd_procovf;     /* # of times the proc table overflowed */
_T_LONG_T       psd_inodeovf;    /* # of times the inode table overflowed */
_T_LONG_T       psd_fileovf;     /* # of times the file table overflowed */
_T_LONG_T       psd_global_virtual; /* Available global virtual space
                                     * (pages) */
int32_t         psd_valid;       /* This is a vector that will indicate
                                  * if a certain field in is valid. */
_T_LONG_T       psd_monarch_node; /* monarch logical node ID      */
_T_LONG_T       psd_node_cnt;    /* number of nodes on the system*/
_T_LONG_T       psd_dnlc_size;   /* Size of DNLC */
_T_ULONG_T      psd_dnlc_hits;   /* DNLC Cache hits */
_T_ULONG_T      psd_dnlc_misses; /* DNLC Cache misses */
_T_ULONG_T      psd_dnlc_long;   /* DNLC Long names tried to lookup */
```

The following fields are obsolete and provided only for backward compatibility: psd_cpu_time, psd_mp_avg_1_min[], psd_mp_avg_5_min[], psd_mp_avg_15_min[], psd_mp_cpu_time[][]. Since the size of these arrays is hard-wired, there is no way to get information about more than a fixed number of processors with the pstat_getdynamic() call. Instead, pstat_getprocessor() should be used. It returns the same information and supports any number of processors.

Macros for field validity check for struct pst_dynamic (member psd_valid) are:
PSD_VM → 0x1
PSD_AVM → 0x2
These bits will be cleared in psd_valid if the corresponding members psd_vm and psd_avm overflow.

pst_fileinfo2

This structure describes per-file information. Each structure returned describes one open file a process. Header file: <sys/pstat/vfs_pstat_body.h>

```
int32_t  psf_valid;             /* Valid vector */
int32_t  psf_ftype;             /* File type, PS_TYPE_VNODE etc. */
pst_subtype_t psf_subtype;      /* File subtype PS_SUBTYPE_CHARDEV ...*/
int32_t  psf_flag;              /* Flags associated with file status */
struct __pst_fid psf_fid;       /* An efficient means to re-access the vnode
                                 * of opened files. */
uint32_t psf_hi_fileid;         /* Per shared file ID */
uint32_t psf_lo_fileid;
uint32_t psf_hi_nodeid;         /* Per vnode/socket ID */
uint32_t psf_lo_nodeid;
int32_t  psf_nstrentt;          /*
                                 * # of entities in a stream. This member is
                                 * valid only for streams or sockets that use
                                 * a stream. Head + Modules + Driver */
_T_LONG_T psf_count;            /* Reference count */
_T_ULONG_T psf_fd;              /*
                                 * File descriptor of the file
                                 * and index for further calls
                                 */
struct __psfileid psf_id;       /* Unique identification of the file */
off32_t     psf_offset;         /* Current 32-bit offset in the file */
```

```
_T_OFF64_T  _PSF_OFFSET64;      /*
                                 * Current 64-bit offset in the file
                                 * If __STDC_32_MODE__, defined as
                                 * off32_t psf_dummy[2];
                                 */
```

This structure provides three types of identifiers, namely: psf_id, psf_fid, and fileid and nodeid. This section explains the usage of these ids. The pstat_getfile2() call returns a list of all open files for a process. The pstat_getproc() call provides the text file, current working directory (cwd), and root. The pstat_getprocvm() call provides the memory regions for the process. If more information is needed for an opened file, a second set of calls, namely: pstat_getfiledetails(), pstat_getsocket(), or pstat_getstream(), or pstat_getpathname(), can be used.

The members hi_fileid, lo_fileid, hi_nodeid, lo_nodeid fields are returned by both sets of calls (with exception of pstat_getpathname()). The user should compare the ids returned from both calls to make sure that the state of the system has not changed. That is, between the pstat_getfile2() call and pstat_getfiledetails() call, the opened file may have been closed. It is possible that the same file descriptor is reassigned to some other file. But fileid and nodeid fields would be different. This detects the state change.

The psf_fid is returned only by the first set of calls, that is pstat_getfile2(), pstat_getproc() and pstat_getprocvm(). psf_fid is a unique ID that is used by kernel to easily access the opened file. Its contents are opaque to the user; they are for kernel use only. The user need only pass the pst_fid back to kernel in the second set of calls so that kernel can efficiently and effectively reaccess the file in question.

The psfsid is returned with every file, cwd, memory regions, and root. This contains the file system identification and inode numbers. This needs to be passed to pstat_getmpathname() call to get all the DNLC entries for that file system.

__psfileid

This structure is an abstraction of a unique identification for an opened file. Header file: <sys/pstat/pstat_body.h>

```
struct __psfsid psf_fsid;       /* Filesystem identification */
_T_LONG_T       psf_fileid;     /* File identification within FS */
_T_LONG_T       psf_spare;      /* Reserved for future expansion */
```

__psfsid

This strucutre is an abstraction of a unique identification for a file system. Header file: <sys/pstat/pstat_body.h>

```
_T_LONG_T psfs_id;      /* Filesystem ID */
_T_LONG_T psfs_type;    /* Filesystem type */
```

pst_filedetails

Header file: <sys/pstat/filedetails_pstat_body.h>

```
uint32_t        psfd_hi_fileid; /* Per shared file ID. See above. */
uint32_t        psfd_lo_fileid;
uint32_t        psfd_hi_nodeid; /* Per vnode ID. See above. */
uint32_t        psfd_lo_nodeid;
```

```
int32_t         psfd_ftype;     /* File type, PS_TYPE_VNODE etc. */
pst_subtype_t   psfd_subtype;   /* File sub type PS_SUBTYPE_CHARDEV etc. */
int32_t         psfd_lckflag;   /* Flags associated with file locking */
/*
 * This is a vector that will indicate if a certain field in the structure is
 * valid or not. User should check this field when EOVERFLOW is set.
 */
int32_t         psfd_valid;     /* PSFD_SIZE → 0x1 */
/* ID of device containing a directory entry for this file */
_T_LONG_T       psfd_dev;
_T_LONG_T       psfd_ino;       /* file Inode number */
_T_ULONG_T      psfd_mode;      /* file type, attributes, and acl */
_T_LONG_T       psfd_nlink;     /* number of links to the file */
_T_LONG_T       psfd_uid;       /* user ID of file owner */
_T_LONG_T       psfd_gid;       /* group ID of file group */
_T_LONG_T       psfd_rdev;      /* device ID (char/block special files*/
_T_LONG_T       psfd_size;      /* file size in bytes */
_T_LONG_T       psfd_atime;     /* time of last access */
_T_LONG_T       psfd_mtime;     /* time of last data modification */
_T_LONG_T       psfd_ctime;     /* time of last file status change */
_T_LONG_T       psfd_blksize;   /* preferred I/O block size */
/* Number of File system specific blocks allocated to this file. */
_T_LONG_T       psfd_blocks;
```

pst_ipcinfo

This structure describes the system-wide global System V IPC constants. These are typically (currently) defined at boot time, but may become dynamic in future releases. Header file: <sys/pstat/ipc_pstat_body.h>

```
_T_LONG_T   psi_semmap; /* resource map size for SysV semaphores */
_T_LONG_T   psi_semmni; /* number of identifiers for SysV semaphoe sets */
_T_LONG_T   psi_semmns; /* system-wide total of SysV semapores */
_T_LONG_T   psi_semmnu; /* system-wide total of SysV sema undo structs */
_T_LONG_T   psi_semmsl; /* max # of semaphores per identifier */
_T_LONG_T   psi_semopm; /* max # of operations per semop() call */
_T_LONG_T   psi_semume; /* max # of undo entries per process */
_T_LONG_T   psi_semusz; /* size in bytes of undo structure */
_T_LONG_T   psi_semvmx; /* maximum value for semaphore */
_T_LONG_T   psi_semaem; /* adjust-on-exit maximum value */
_T_LONG_T   psi_msgmap; /* resource map size for SysV messages */
_T_LONG_T   psi_msgmax; /* maximum message size */
_T_LONG_T   psi_msgmnb; /* maximum bytes on message queue */
_T_LONG_T   psi_msgmni; /* system-wide total of SysV msg queue IDs */
_T_LONG_T   psi_msgssz; /* message segment size */
_T_LONG_T   psi_msgtql; /* system-wide total of SysV msg headers */
_T_LONG_T   psi_msgseg; /* system-wide total of SysV msg segments */
_T_LONG_T   psi_shmmax; /* maximum shared memory segment size */
_T_LONG_T   psi_shmmin; /* minimum shared memory segment size */
_T_LONG_T   psi_shmmni; /* system-wide total of SysV shm identifiers */
_T_LONG_T   psi_shmseg; /* max # of attached SysV shm segs per process */
int32_t     psi_valid;  /* valid vector. PSI_SHMMAX → 0x1 */
```

pst_lvinfo

This structure contains per-logical volume information. Each structure returned describes one logical volume. Header file <sys/pstat/lv_pstat_body.h>

```
_T_ULONG_T      psl_idx;        /* Index for further pstat() requests */
struct __psdev  psl_dev;        /* device specification for the volume */
_T_ULONG_T      psl_rxfer;      /* # of reads */
_T_ULONG_T      psl_rcount;     /* # of bytes read */
_T_ULONG_T      psl_wxfer;      /* # of writes */
_T_ULONG_T      psl_wcount;     /* # of bytes written */
_T_ULONG_T      psl_openlv;     /* # of opened LV's in this LV's LVG */
_T_ULONG_T      psl_mwcwaitq;   /* Length of LV's mirror write consistency
                                 * Cache (MWC) */
_T_ULONG_T      psl_mwcsize;    /* Size of LV's LVG's MWC */
_T_ULONG_T      psl_mwchits;    /* # of hits to the LV's LVG's MWC */
_T_ULONG_T      psl_mwcmisses;  /* # of misses to the LV's LVG's MWC */
```

lwp_status

This structure contains lightweight process (LWP) or thread information. This is only available as 64-bit data. Header file: <sys/pstat/lwp_pstat_body.h>

```
int64_t     lwp_idx;        /* Index for further pstat_getlwp() calls */
int64_t     lwp_lwpid;      /* LWP ID */
int64_t     lwp_pid;        /* PID that LWP belongs to */
int64_t     lwp_flag;       /* flags associated with LWP */
int64_t     lwp_stat;       /* Current status */
int64_t     lwp_wchan;      /* If state LWP_SLEEP, value sleeping on */
int64_t     lwp_pri;        /* priority of LWP */
int64_t     lwp_cpu;        /* (decaying) CPU utilization for scheduling */
int64_t     lwp_spu;        /* spu number LWP is assigned to */
int64_t     lwp_user_suspcnt;/* user-initiated suspend count */
struct pstsigset lwp_sig;    /* signals pending to LWP */
struct pstsigset lwp_sigmask;/* current signal mask */
int64_t     lwp_schedpolicy; /* scheduling policy */
int64_t     lwp_ticksleft;   /* clock ticks left in LWP's RR time slice */
int64_t     lwp_start;       /* time LWP created (seconds since epoch) */
uint64_t    lwp_minorfaults; /* # page reclaims */
uint64_t    lwp_majorfaults; /* # page faults needing disk access */
uint64_t    lwp_ndeact;      /* # deactivates */
uint64_t    lwp_inblock;     /* # block input operations */
uint64_t    lwp_oublock;     /* # block output operations */
uint64_t    lwp_ioch;        /* # of characters read/written */
uint64_t    lwp_msgsnd;      /* # messages sent */
uint64_t    lwp_msgrcv;       /* # messages received */
uint64_t    lwp_nsignals;     /* # signals received */
uint64_t    lwp_nvcsw;        /* # voluntary context switches */
uint64_t    lwp_nivcsw;       /* # involuntary context switches */
uint64_t    lwp_syscall;      /* # syscalls */
int64_t     lwp_syscall_code; /* last syscall code */
int64_t     lwp_utime;        /* user time spent executing (in seconds) */
int64_t     lwp_stime;        /* system time spent executing (in seconds) */
struct lwpcycles lwp_usercycles;      /* 64-bit user mode execution cycle
                                       * count */
struct lwpcycles lwp_systemcycles;    /* 64-bit system mode execution cycle
                                       * count */
struct lwpcycles lwp_interruptcycles; /* 64-bit interrupt for thread cycle
                                       * count */
int64_t     lwp_valid;        /* valid vector */
```

```
int64_t     lwp_fss;          /* fair share scheduler group ID */
int64_t     lwp_bind_flags;   /* processor/ldom binding flag */
int64_t     lwp_bind_spu;     /* processor bound to */
int64_t     lwp_bind_ldom;    /* ldom bound to */
int64_t     lwp_reserved1;    /* reserved  - do not use */
int64_t     lwp_reserved2;    /* reserved  - do not use */
int64_t     lwp_reserved3;    /* reserved  - do not use */
```

pst_mpathnode

Header file: <sys/pstat/rpath_pstat_body.h>

```
_T_ULONG_T psr_idx;     /*
                         * Current index of the entry on the chain of DNLC
                         * entries. Index for further pstat_getmpathname
                         * calls.
                         */
struct __psfileid psr_file;     /* ID of the file this entry describes */
struct __psfileid psr_parent;   /* ID of the parent of this file */
char       psr_name[PS_SEGMENTNAME_SZ];  /* NULL terminated name of entry */
```

The segment name size is: PS_SEGMENTNAME_SZ → 64

pst_msginfo

This structure describes per System V message queue information. Header file: <sys/pstat/ipc_pstat_body.h>

```
_T_ULONG_T      psm_idx;        /* Idx for further pstat() requests */
_T_LONG_T       psm_uid;        /* UID of msg queue owner */
_T_LONG_T       psm_gid;        /* GID of msg queue owner */
_T_LONG_T       psm_cuid;       /* UID of msg queue creator */
_T_LONG_T       psm_cgid;       /* GID of msg queue creator */
_T_ULONG_T      psm_mode;       /* mode of msg queue (9 bits) */
_T_ULONG_T      psm_seq;        /* sequence number of msg queue */
_T_ULONG_T      psm_key;        /* IPC key of msg queue */
_T_ULONG_T      psm_qnum;       /* number of msgs on this queue */
_T_ULONG_T      psm_qbytes;     /* max bytes for msgs on this queue */
_T_ULONG_T      psm_cbytes;     /* cur bytes for msgs on this queue */
_T_LONG_T       psm_lspid;      /* PID of last msgsnd()er */
_T_LONG_T       psm_lrpid;      /* PID of last msgrcv()er */
_T_LONG_T       psm_stime;      /* last msgsnd() time (since 1970) */
_T_LONG_T       psm_rtime;      /* last msgrcv() time (since 1970) */
_T_LONG_T       psm_ctime;      /* last change time (since 1970) */
_T_ULONG_T      psm_flags;      /* flags for the message queue */
```

Definitions of flag bits in psm_flags:

PS_MSG_ALLOC → 0x1 /* message queue is in use */

PS_MSG_RWAIT → 0x2 /* one or more processes waiting to read */

PS_MSG_WWAIT → 0x4 /* one or more processes waiting to write */

pst_node

This structure describes SCA (ccNUMA) node information. It is only available as 64-bit data (_PSTAT64 defined). Header file: <sys/pstat/node_pstat_body.h>

```
uint64_t        psn_idx;          /* value currently undefined  */
uint64_t        psn_logical_node; /* logical node ID            */
uint64_t        psn_physical_node;/* physical node ID           */
uint64_t        psn_cpu_cnt;      /* number of node active CPUs */
uint64_t        psn_private_mem_size; /* node private memory in (MB)*/
uint64_t        psn_gmem_start;   /* start of node global memory(in MB) */
uint64_t        psn_gmem_size;    /* max size of node global memory(in MB) */
uint64_t        psn_reserved[3];  /* Reserved for future use    */
uint64_t        psn_padding[6];   /* Padding to next pow(2)     */
```

pst_status

This structure contains per-process information. Header file: <sys/pstat/pm_pstat_body.h>

```
_T_LONG_T pst_idx;      /* Index for further pstat() requests */
_T_LONG_T pst_uid;      /* Real UID */
_T_LONG_T pst_pid;      /* Process ID */
_T_LONG_T pst_ppid;     /* Parent process ID */
_T_LONG_T pst_dsize;    /* # real pages used for data */
_T_LONG_T pst_tsize;    /* # real pages used for text */
_T_LONG_T pst_ssize;    /* # real pages used for stack */
_T_LONG_T pst_nice;     /* Nice value */
struct __psdev pst_term;/* TTY of this process; -1/-1 if there isn't one */
_T_LONG_T pst_pgrp;     /* Process group of this process */
_T_LONG_T pst_pri;      /* priority of process */
_T_LONG_T pst_addr;     /* address of process (in memory) */
_T_LONG_T pst_cpu;      /* processor utilization for scheduling */
_T_LONG_T pst_utime;    /* user time spent executing (in seconds) */
_T_LONG_T pst_stime;    /* system time spent executing (in seconds) */
_T_LONG_T pst_start;    /* time process started (seconds since epoch) */
_T_LONG_T pst_flag;     /* flags associated with process */
_T_LONG_T pst_stat;     /* Current status */
_T_LONG_T pst_wchan;    /* If state PS_SLEEP, value sleeping on */
_T_LONG_T pst_procnum;  /* processor this proc last run on */
char pst_cmd[PST_CLEN]; /* Command line for the process, if available */
_T_LONG_T pst_time;     /* resident time for scheduling */
_T_LONG_T pst_cpticks;  /* ticks of CPU time */
_T_LONG_T pst_cptickstotal; /* total ticks for life of process */
_T_LONG_T pst_fss;      /* fair share scheduler group ID */
float pst_pctcpu;       /* %CPU for this process during p_time */
_T_LONG_T pst_rssize;   /* resident set size for process (private pages)*/
_T_LONG_T pst_suid;     /* saved UID */
char pst_ucomm[PST_UCOMMLEN]; /* executable basename the process is running*/
_T_LONG_T pst_shmsize;  /* # real pages used for shared memory */
_T_LONG_T pst_mmsize;   /* # real pages used for memory mapped files */
_T_LONG_T pst_usize;    /* # real pages used for U-Area & K-Stack */
_T_LONG_T pst_iosize;   /* # real pages used for I/O device mapping */
_T_LONG_T pst_vtsize;   /* # virtual pages used for text */
_T_LONG_T pst_vdsize;   /* # virtual pages used for data */
_T_LONG_T pst_vssize;   /* # virtual pages used for stack */
_T_LONG_T pst_vshmsize; /* # virtual pages used for shared memory */
_T_LONG_T pst_vmmsize;  /* # virtual pages used for mem-mapped files */
_T_LONG_T pst_vusize;   /* # virtual pages used for U-Area & K-Stack */
_T_LONG_T pst_viosize;  /* # virtual pages used for I/O dev mapping */
_T_ULONG_T pst_minorfaults; /* # page reclaims for the process */
_T_ULONG_T pst_majorfaults; /* # page faults needing disk access */
_T_ULONG_T pst_nswap;   /* # of swaps for the process */
```

```
_T_ULONG_T pst_nsignals;/* # signals received by the process */
_T_ULONG_T pst_msgrcv;  /* # socket msgs received by the proc*/
_T_ULONG_T pst_msgsnd;  /* # of socket msgs sent by the proc */
_T_LONG_T pst_maxrss;   /* highwater mark for proc resident set size */
_T_LONG_T pst_sid;      /* session ID */
_T_LONG_T pst_schedpolicy;  /* scheduling policy for the process */
_T_LONG_T pst_tickslеft;    /* clock ticks left in process' RR timeslice */
struct __psfileid pst_rdir; /* File ID of the process' root directory */
struct __psfileid pst_cdir; /* File ID of the process' current directory */
struct __psfileid pst_text; /* File ID of the process' executable */
_T_LONG_T pst_highestfd;    /* highest file descriptor currently opened */
_T_LONG_T pst_euid;     /* Effective UID */
_T_LONG_T pst_egid;     /* Effective GID */
_T_LONG_T pst_ioch;     /* # of characters read/written */
struct __pstcycles pst_usercycles;/* 64-bit user mode execution cycle count*/
struct __pstcycles pst_systemcycles; /* 64-bit system mode execution cycle
                                      * count */
struct __pstcycles pst_interruptcycles;  /* 64-bit interrupt for process
                                          * cycle count */
_T_LONG_T pst_gid;      /* Real GID */
_T_LONG_T pst_sgid;     /* saved effective gid */
_T_LONG_T pst_nlwps;    /* # LWPs within this process */
struct pstsigset pst_psig;      /* signals pending to proc */
_T_LONG_T pst_lwpid;  /* LWP identifier. NOTE: If this process multi-
                       * threaded, this is an lwpid of one of LWPs in the
                       * process at this exact moment, which LWP is undefined
                       * (random)*/
int32_t pst_valid;      /* valid vector */
_T_LONG_T pst_text_size;/* Page size used for text objects. */
_T_LONG_T pst_data_size;/* Page size used for data objects. */
struct __pstcycles pst_child_usercycles;  /* dead children user mode
                                           execution cycle count */
struct __pstcycles pst_child_systemcycles;/* dead children system mode
                                           * execution cycle count */
struct __pstcycles pst_child_interruptcycles;  /* dead children interrupt
                                                * mode execution cycle count */
struct __psttime pst_child_utime; /* reaped child user mode execution time */
struct __psttime pst_child_stime; /*reaped child system mode execution time*/
_T_LONG_T pst_inblock;  /* block input operations */
_T_LONG_T pst_oublock;  /* block output operations */
_T_LONG_T pst_nvcsw;    /* voluntary context switches */
_T_LONG_T pst_nivcsw;   /* involuntary context switches */
_T_LONG_T pst_child_inblock;   /* reaped child block input operations */
_T_LONG_T pst_child_oublock;   /* reaped child block output operations */
_T_LONG_T pst_child_ioch;      /* reaped child # of chars read/written */
_T_LONG_T pst_child_msgsnd;    /* reaped child # of messages sent */
_T_LONG_T pst_child_msgrcv;    /* reaped child # of messages received */
_T_LONG_T pst_child_nvcsw;     /* reaped child voluntary context switches */
_T_LONG_T pst_child_nivcsw;    /* reaped child involuntary context switches*/
_T_LONG_T pst_child_minorfaults;    /* reaped child # of page reclaims */
_T_LONG_T pst_child_majorfaults;    /* reaped child # of page faults
                                     * needing disk access */
_T_LONG_T pst_logical_node;   /* node this proc last run on   */
uint32_t pst_hi_fileid_cdir;  /* per shared file ID */
uint32_t pst_lo_fileid_cdir;
uint32_t pst_hi_nodeid_cdir;  /* per vnode ID */
uint32_t pst_lo_nodeid_cdir;
```

```
struct __pst_fid pst_fid_cdir;/* Cookie for current working directory */
uint32_t pst_hi_fileid_rdir;  /* per shared file ID */
uint32_t pst_lo_fileid_rdir;
uint32_t pst_hi_nodeid_rdir;  /* per vnode ID */
uint32_t pst_lo_nodeid_rdir;
struct __pst_fid pst_fid_rdir;/* Cookie for root directory */
uint32_t pst_hi_fileid_text;  /* per shared file ID */
uint32_t pst_lo_fileid_text;
uint32_t pst_hi_nodeid_text;  /* per vnode ID */
uint32_t pst_lo_nodeid_text;
struct __pst_fid pst_fid_text;/* Cookie for text file */
```

Lengths for cached command line and u_comm entries:
PST_CLEN → 64
PST_UCOMMLEN → (14 + 1)

pst_processor
This structure describes per-processor information. Header file: <sys/pstat/global_pstat_body.h>

```
_T_ULONG_T      psp_idx;        /* Index of the current spu in the array
                                 * of processor statistic entries. */
_T_ULONG_T      psp_fsreads;    /* # of reads from filesys blocks. */
_T_ULONG_T      psp_fswrites;   /* # of writes to filesys blocks. */
_T_ULONG_T      psp_nfsreads;   /* # of NFS disk blk reads issued. */
_T_ULONG_T      psp_nfswrites;  /* # of NFS disk blk writes issued. */
_T_ULONG_T      psp_bnfsread;   /* # of bytes read from NFS. */
_T_ULONG_T      psp_bnfswrite;  /* # of bytes written to NFS. */
_T_ULONG_T      psp_phread;     /* # of physical reads to raw devs. */
_T_ULONG_T      psp_phwrite;    /* # of physical writes to raw devs. */
_T_ULONG_T      psp_runocc;     /* # of times the processor had processes
                                 * waiting to run.  This running total is
                                 * updated once a second. */
_T_ULONG_T      psp_runque;     /* # of processes the processor had waiting
                                 * to run.  This running total is updated
                                 * once a second. */
_T_ULONG_T      psp_sysexec;    /* # of exec system calls. */
_T_ULONG_T      psp_sysread;    /* # of read system calls. */
_T_ULONG_T      psp_syswrite;   /* # of write system calls. */
_T_ULONG_T      psp_sysnami;    /* # of calls to sysnami(). */
_T_ULONG_T      psp_sysiget;    /* # of calls to sysiget(). */
_T_ULONG_T      psp_dirblk;     /* # of filesystem blocks read doing
                                 * directory lookup. */
_T_ULONG_T      psp_semacnt;    /* # of System V semaphore ops. */
_T_ULONG_T      psp_msgcnt;     /* # of System V message ops. */
_T_ULONG_T      psp_muxincnt;   /* # of MUX interrupts received. */
_T_ULONG_T      psp_muxoutcnt;  /* # of MUX interrupts sent. */
_T_ULONG_T      psp_ttyrawcnt;  /* # of raw characters read. */
_T_ULONG_T      psp_ttycanoncnt; /* # of canonical chars processed. */
_T_ULONG_T      psp_ttyoutcnt;  /* # of characters output. */
struct __pscoproc psp_coprocessor; /* info on any co-processors */
_T_ULONG_T      psp_iticksperclktick; /* interval timer counts (CR16) per
                                       * clock tick, see sysconf(_SC_CLK_TCK)*/
_T_ULONG_T      psp_sysselect;  /* # of select system calls. */
struct __pstcycles psp_idlecycles; /* 64-bit idle execution cycle count */
double          psp_avg_1_min;  /* per-processor run queue lengths */
```

```
double          psp_avg_5_min;
double          psp_avg_15_min;
/* per-processor CPU time/state */
_T_LONG_T       psp_cpu_time[PST_MAX_CPUSTATES];
_T_ULONG_T      psp_logical_node; /* node the spu is on */
```

pst_vm_status

This structure contains process address space information. Header file: <vm_pstat_body.h>

```
_T_LONG_T   pst_space;      /* virtual space for region */
_T_LONG_T   pst_vaddr64bit_pad;
_T_LONG_T   pst_vaddr;      /* virtual offset for region */
_T_LONG_T   pst_length;     /* number of pages mapped by pregion */
_T_LONG_T   pst_phys_pages; /* number of valid pages in region */
_T_LONG_T   pst_flags;
_T_LONG_T   pst_type;
_T_LONG_T   pst_permission; /* protection type (not protid) of region */
struct __psfileid pst_id;
int32_t     pst_valid;      /* valid vector. PST_VADDR → 0x1 */
_T_LONG_T   pst_pagesize_hint;
_T_LONG_T   pst_vps_pgsizes[PST_N_PG_SIZES];
struct __pst_fid  pst_fid;  /* A unique ID to efficiently reaccess the file
                             * corresponding to the memory region */
_T_ULONG_T  pst_refcnt;     /* # of processes sharing the region */
_T_ULONG_T  pst_incore;     /* # of non-swapped processes sharing region */
_T_LONG_T   pst_lockmem;    /* # of pages locked in memory  */
uint32_t    pst_hi_fileid;  /* Per shared file ID */
uint32_t    pst_lo_fileid;
uint32_t    pst_hi_nodeid;  /* Per vnode ID*/
uint32_t    pst_lo_nodeid;
uint32_t    padding;
```

Number of page sizes: PST_N_PG_SIZES → 16

pst_seminfo

This structure describes System V semaphore set information. Header file: <sys/pstat/ipc_pstat_body.h>

```
_T_ULONG_T      pse_idx;        /* Index for further pstat() requests */
_T_LONG_T       pse_uid;        /* UID of semaphore set owner */
_T_LONG_T       pse_gid;        /* GID of semaphore set owner */
_T_LONG_T       pse_cuid;       /* UID of semaphore set creator */
_T_LONG_T       pse_cgid;       /* GID of semaphore set creator */
_T_ULONG_T      pse_mode;       /* mode of semaphore set (9 bits) */
_T_ULONG_T      pse_seq;        /* sequence number of semaphore set */
_T_ULONG_T      pse_key;        /* IPC key of semaphore set */
_T_ULONG_T      pse_nsems;      /* number of semaphores for this set */
_T_LONG_T       pse_otime;      /* last semop time (secs since 1970) */
_T_LONG_T       pse_ctime;      /* last change time (since 1970) */
_T_ULONG_T      pse_flags;      /* flags for the semaphore set */
```

Flag bit definitions for pse_flags: PS_SEM_ALLOC → 0x1 /* semaphore set is in use */

pst_shminfo

This structure describes System V shared memory segment) information. Each structure returned describes on segment identifier on the system. Header file: <sys/pstat/ipc_pstat_body.h>

```
_T_ULONG_T      psh_idx;        /* Index for further pstat() requests */
_T_LONG_T       psh_uid;        /* UID of shm segment owner */
_T_LONG_T       psh_gid;        /* GID of shm segment owner */
_T_LONG_T       psh_cuid;       /* UID of shm segment creator */
_T_LONG_T       psh_cgid;       /* GID of shm segment creator */
_T_ULONG_T      psh_mode;       /* mode of shm segment (9 bits) */
_T_ULONG_T      psh_seq;        /* sequence number of shm segment */
_T_ULONG_T      psh_key;        /* IPC key of shm segment ID */
_T_ULONG_T      psh_segsz;      /* size of shm segment (bytes) */
_T_LONG_T       psh_cpid;       /* PID of shm segment creator */
_T_LONG_T       psh_lpid;       /* PID of last shmop() */
_T_ULONG_T      psh_nattch;     /* current # of procs attached (accurate)*/
_T_ULONG_T      psh_cnattch;    /* current # attached/in mem (inaccurate)*/
_T_LONG_T       psh_atime;      /* last shmat() time (since 1970) */
_T_LONG_T       psh_dtime;      /* last shmdt() time (since 1970) */
_T_LONG_T       psh_ctime;      /* last change time (since 1970) */
_T_ULONG_T      psh_flags;      /* flags for the shm segment */
int32_t         psh_valid;      /* Valid vector. PSH_SEGSZ → 0x1 */
```

Flag bit definitions for psh_flags:

PS_SHM_ALLOC → 0x1 /* shared memory segment is in use */

PS_SHM_DEST → 0x2 /* shm segment to be deleted on last detach */

PS_SHM_CLEAR →0x4 /* shm segment to be zeroed on first attach */

pst_socket

Header file: <sys/pstat/socket_pstat_body.h>

```
uint32_t    pst_hi_fileid;      /* File ID */
uint32_t    pst_lo_fileid;
uint32_t    pst_hi_nodeid;      /* Socket node ID */
uint32_t    pst_lo_nodeid;
uint32_t    pst_peer_hi_nodeid; /* Peer Socket node ID; for AF_UNIX only */
uint32_t    pst_peer_lo_nodeid;
_T_LONG_T   pst_flags;          /* Is a stream? */
uint32_t    pst_type;           /* generic type */
uint32_t    pst_options;        /* Option flags per socket */
uint32_t    pst_linger;         /* time to linger while closing */
uint32_t    pst_state;          /* Internal socket state */
uint32_t    pst_family;         /* domain protocol; a member of */
uint32_t    pst_protocol;       /* protocol number for this family(such as
                                 * IPPROTO_*), if applicable */
uint32_t    pst_qlimit;         /* max #of queued connections */
uint32_t    pst_qlen;           /* # connections on incoming q ready to be
                                 * accepted*/
_T_ULONG_T  pst_idata;          /* actual chars in inbound buffer */
uint32_t    pst_ibufsz;         /* max actual char count */
_T_ULONG_T  pst_rwnd;           /* advertised rcv window, if applicable */
_T_ULONG_T  pst_odata;          /* actual chars in outbound buffer */
uint32_t    pst_obufsz;         /* max actual char count */
_T_ULONG_T  pst_swnd;           /* current send window, if applicable */
```

```
uint32_t    pst_pstate;         /* protocol state */
_T_ULONG_T  pst_boundaddr_len;  /* Length of pst_boundaddr */
_T_ULONG_T  pst_remaddr_len;    /* Length of pst_remaddr */
char        pst_boundaddr[PS_ADDR_SZ]; /* local addr/port, etc */
char        pst_remaddr[PS_ADDR_SZ];   /* peer addr/port, etc */
```

The members pst_boundaddr and pst_remaddr contain data of the form struct sockaddr, sockaddr_un, sockaddr_in, sockaddr_in6, etc. depending on the socket family. In case of AF_UNIX sockets, pst_peer_hi_nodeid and pst_peer_lo_nodeid fields can be used to find the peer socket by matching them with pst_hi_nodeid and pst_lo_nodeid of the other sockets.

PS_ADDR_SZ → 256

pst_stable

This structure describes the information in any system's stable storage area. Header file: <sys/pstat/mdep_pstat_body.h>

```
_T_LONG_T       pss_size;       /* Number of bytes of stable store */
_T_LONG_T       pss_type;       /* Type of stable store */
_T_LONG_T       pss_pad[8];     /* Reserved */
union {
      unsigned char Pss_PA_RISC_buffer[PST_MAX_PA_RISC_STABLE];
                                      /* Uninterpreted stable store data */
      struct pst_stable_PA_RISC Pss_PA_RISC_stable;
                                      /* PA-RISC stable store data */
} pss_un;
```

For more information, refer to mdep_pstat_body.h file.
Values for pss_type: PS_PA_RISC → 0x1 /* PA-RISC stable store */
Max bytes in a PA-RISC stable store area: PST_MAX_PA_RISC_STABLE → 256

pst_static

This structure contains static system information -- data that will remain the same (at least) until reboot. Header file: <sys/pstat/global_pstat_body.h>

```
_T_LONG_T       max_proc;           /* Number of process table entries */
struct __psdev  console_device;     /* console major & minor number */
_T_LONG_T       boot_time;
_T_LONG_T       physical_memory;    /* system physical memory in 4K pages */
_T_LONG_T       page_size;          /* bytes/page */
_T_LONG_T       cpu_states;         /* Number of CPU states */
_T_LONG_T       pst_status_size;    /* Size of data structure pst_status */
_T_LONG_T       pst_static_size;
_T_LONG_T       pst_dynamic_size;
_T_LONG_T       pst_vminfo_size;
_T_LONG_T       command_length;     /* length for cached command line */
_T_LONG_T       pst_processor_size; /* size of data structure pst_processor*/
_T_LONG_T       pst_diskinfo_size;
_T_LONG_T       pst_lvinfo_size;
_T_LONG_T       pst_swapinfo_size;
_T_LONG_T       pst_maxmem;         /* actual max memory per process */
_T_LONG_T       pst_lotsfree;       /* Vhand tunables. max free before clock
```

```
                                    * freezes */
_T_LONG_T       pst_desfree;        /* # of pages to try to keep free via
                                    * daemon */
_T_LONG_T       pst_minfree;        /* minimum free pages before swapping
                                    * begins */
_T_LONG_T       pst_max_ninode;     /* Max number of Inodes */
_T_LONG_T       pst_max_nfile;      /* Max number of file table entries */
_T_LONG_T       pst_stable_size;    /* size of data structure pst_stable */
_T_LONG_T       pst_supported_pgsize[PST_N_PG_SIZES]; /* valid page sizes
                                        * supported by the architecture */
_T_LONG_T       pst_fileinfo_size;  /* size of data structure pst_fileinfo */
_T_LONG_T       pst_fileinfo2_size;
_T_LONG_T       pst_filedetails_size;
_T_LONG_T       pst_socket_size;
_T_LONG_T       pst_stream_size;
_T_LONG_T       pst_mpathnode_size;
_T_LONG_T       pst_ipcinfo_size;
_T_LONG_T       pst_msginfo_size;
_T_LONG_T       pst_seminfo_size;
_T_LONG_T       pst_shminfo_size;
_T_LONG_T       pst_vm_status_size;
_T_LONG_T       lwp_status_size;
_T_LONG_T       pst_crashinfo_size;
_T_LONG_T       pst_crashdev_size;
_T_LONG_T       pst_node_size;
_T_LONG_T       clonemajor;         /* Major number of clone devices */
```

Number of page sizes: PST_N_PG_SIZES → 16

pst_stream:

Header file: <sys/pstat/stream_pstat_body.h>

```
pst_streamentity_t type;      /* PS_STR_HEAD, PS_STR_MODULE, PS_STR_DRIVER */
union {
      struct {                /* head Information */
            uint32_t pst_hi_fileid; /* Per file ID*/
            uint32_t pst_lo_fileid;
            uint32_t pst_hi_nodeid; /* Per node ID*/
            uint32_t pst_lo_nodeid;
            _T_ULONG_T pst_rbytes;  /* queued bytes on read side */
            _T_ULONG_T pst_wbytes;  /* queued bytes on write side */
            _T_ULONG_T pst_flag;    /* Is a clone? PS_STR_ISACLONE*/
            _T_LONG_T pst_dev_major;/* major number */
            _T_LONG_T pst_dev_minor;/* minor number */
            _T_LONG_T pst_dev_seq;  /* clone driver sequence */
      } head;
      struct { /* Module Information */
            _T_ULONG_T pst_rbytes; /* queued bytes on read side */
            _T_ULONG_T pst_wbytes; /* queued bytes on write side */
            char pst_name[PS_STRMODNAME_SZ]; /* NULL terminated */
      } module;
      struct { /* driver information */
            _T_ULONG_T pst_rbytes; /* queued bytes on read side */
            _T_ULONG_T pst_wbytes; /* queued bytes on write side */
            char pst_name[PS_STRMODNAME_SZ]; /* NULL terminated */
      } driver;
```

```
} val;
```

Maximum module name length: PS_STRMODNAME_SZ → 32

The type field pst_stream structure can be PS_STR_HEAD, PS_STR_MODULE or PS_STR_DRIVER. The union val in pst_stream will represent the structures head, module, or driver in the respective cases. If the flag PS_STR_ISACLONE is set in pst_flag for head, the field pst_dev_seq in head represents the clone driver sequence number the stream.

pst_swapinfo

This structure describes per-swap-area information. Each structure returned describes one "pool" of swap space on the system, either a block device or a portion of a file system. Header file: <sys/pstat/vm_pstat_body.h>

```
_T_ULONG_T      pss_idx;        /* Index for further pstat() requests */
_T_ULONG_T      pss_flags;      /* flags associated with swap pool */
_T_ULONG_T      pss_priority;   /* priority of the swap pool */
_T_ULONG_T      pss_nfpgs;      /* # of free pages of space in pool */
union {                         /* block and fs swap differ */
      struct __pss_blk Pss_blk;   /* Block device Fields */
      struct __pss_fs  Pss_fs;    /* File System Fields */
      struct pss_reserved Pss_XX; /* reserved for union expansion */
} pss_un;
_T_ULONG_T      pss_swapchunk;  /* block size */
```

For more information, refer to header file vm_pstat_body.h. Flag values for pss_flags:
SW_ENABLED → 0x1
SW_BLOCK → 0x2
SW_FS → 0x4

pst_vminfo

Header file: <sys/pstat/vm_pstat_body.h>

```
_T_LONG_T       psv_rdfree;     /* rate: pages freed by daemon */
_T_LONG_T       psv_rintr;      /* device interrupts */
_T_LONG_T       psv_rpgpgin;    /* pages paged in */
_T_LONG_T       psv_rpgpgout;   /* pages paged out */
_T_LONG_T       psv_rpgrec;     /* total page reclaims */
_T_LONG_T       psv_rpgtlb;     /* tlb flushes - 800 only */
_T_LONG_T       psv_rscan;      /* scans in pageout daemon */
_T_LONG_T       psv_rswtch;     /* context switches */
_T_LONG_T       psv_rsyscall;   /* calls to syscall() */
_T_LONG_T       psv_rxifrec;    /* found in freelist rather than in file
                                 * sys */
_T_LONG_T       psv_rxsfrec;    /* found in freelist rather than on swap
                                 * dev */
_T_LONG_T       psv_cfree;      /* cnt: free memory pages */
_T_LONG_T       psv_sswpin;     /* sum: swapins */
_T_LONG_T       psv_sswpout;    /* swapouts */
_T_LONG_T       psv_sdfree;     /* pages freed by daemon */
_T_LONG_T       psv_sexfod;     /* pages filled on demand from executables */
_T_LONG_T       psv_sfaults;    /* total faults taken */
_T_LONG_T       psv_sintr;      /* device interrupts */
_T_LONG_T       psv_sintrans;   /* in-transit blocking page faults */
_T_LONG_T       psv_snexfod;    /* number of exfod's created */
```

```
_T_LONG_T       psv_snzfod;     /* number of zero filled on demand */
_T_LONG_T       psv_spgfrec;    /* page reclaims from free list */
_T_LONG_T       psv_spgin;      /* pageins */
_T_LONG_T       psv_spgout;     /* pageouts */
_T_LONG_T       psv_spgpgin;    /* pages paged in */
_T_LONG_T       psv_spgpgout;   /* pages paged out */
_T_LONG_T       psv_spswpin;    /* pages swapped in */
_T_LONG_T       psv_spswpout;   /* pages swapped out */
_T_LONG_T       psv_srev;       /* revolutions of the hand */
_T_LONG_T       psv_sseqfree;   /* pages taken from sequential programs */
_T_LONG_T       psv_sswtch;     /* context switches */
_T_LONG_T       psv_ssyscall;   /* calls to syscall() */
_T_LONG_T       psv_strap;      /* calls to trap */
_T_LONG_T       psv_sxifrec;    /* sum: found in free list rather than in
                                 * file sys */
_T_LONG_T       psv_sxsfrec;    /* sum: found on free list rather than on
                                 * swap dev*/
_T_LONG_T       psv_szfod;      /* pages zero filled on demand */
_T_LONG_T       psv_sscan;      /* scans in pageout daemon */
_T_LONG_T       psv_spgrec;     /* total page reclaims */
_T_LONG_T       psv_deficit;    /* estimate of needs of new swapped-in
                                 * procs */
_T_LONG_T       psv_tknin;      /* number of characters read from ttys */
_T_LONG_T       psv_tknout;     /* number of characters written to ttys */
_T_LONG_T       psv_cntfork;    /* number of forks */
_T_LONG_T       psv_sizfork;    /* number of pages forked */
_T_ULONG_T      psv_lreads;     /* number of disk blk reads issued */
_T_ULONG_T      psv_lwrites;    /* number of disk blk writes issued */
_T_ULONG_T      psv_swpocc;     /* # of times swrq occ'd since boot */
_T_ULONG_T      psv_swpque;     /* cumulative len of swrq since boot */
_T_LONG_T       psv_paging_thold;/* paging threshold, moves between
                                    pst_desfree & pst_lotsfree */
_T_LONG_T       psv_sysmem;      /* pages of memory unavailable for
                                    in-memory backing store */
_T_LONG_T       psv_swapspc_cnt; /* pages of on-disk backing store */
_T_LONG_T       psv_swapspc_max; /* max pages of on-disk backing store */
_T_LONG_T       psv_swapmem_cnt; /* pages of in-memory backing store */
_T_LONG_T       psv_swapmem_max; /* max pages of in-memory backing store */
_T_LONG_T       psv_swapper_mem; /* pages of backing store management
                                    overhead:-
                                    psv_swapper_mem + malloc space
                                    = psv_swapmem_cnt */
_T_LONG_T       psv_lreadsize;   /* # of char xfer'd by bread */
_T_LONG_T       psv_lwritesize;  /* # of char xfer'd by bwrite */
_T_LONG_T       psv_swapmem_on;  /* in-memory backing store enabled */

_T_LONG_T psv_select_success[PST_N_PG_SIZES];
                                /* success by page size of LP fault page
                                 * size selection */
_T_LONG_T psv_select_failure[PST_N_PG_SIZES];
                                /* failure by page size of LP fault page
                                 * size selection */
_T_LONG_T psv_pgalloc_success[PST_N_PG_SIZES];
                                /* success by page size of LP allocation */
_T_LONG_T psv_pgalloc_failure[PST_N_PG_SIZES];
                                /* failure by page size of LP allocation */
_T_LONG_T psv_demotions[PST_N_PG_SIZES];
```

```
                                /* LP demotions by page size */
_T_ULONG_T  psv_reserved1;      /* reserved -- do not use */
_T_ULONG_T  psv_reserved2;      /* reserved -- do not use */
_T_ULONG_T  psv_reserved3;   /* reserved -- do not use */
_T_ULONG_T  psv_sbreadcache; /* Total bread cache hits */
_T_ULONG_T  psv_sbreada;     /* Total read aheads */
_T_ULONG_T  psv_sbreadacache;/* Total read ahead cache hits */
_T_ULONG_T  psv_rpswpin;     /* Rate pages swapped in */
_T_ULONG_T  psv_rpswpout;    /* Rate pages swapped out */
_T_ULONG_T  psv_rpgin;       /* Rate pageins */
_T_ULONG_T  psv_rpgout;      /* Rate pageouts */
_T_ULONG_T  psv_rfaults;     /* Rate faults taken */
_T_ULONG_T  psv_rbread;      /* Rate breads */
_T_ULONG_T  psv_rbreadcache; /* Rate bread cache hits */
_T_ULONG_T  psv_rbreada;     /* Rate read aheads */
_T_ULONG_T  psv_rbreadacache;/* Rate read ahead cache hits */
_T_ULONG_T  psv_rswpin;      /* Rate swapins */
_T_ULONG_T  psv_rswpout;     /* Rate swapouts */
_T_LONG_T   psv_kern_dynmem; /*# of pages allocated for kernel dynamic
                              * memory */
```