

# HP-UX Processor Sets

A Technical White Paper

HP-UX 11i



**Manufacturing Part Number: 5185-4322**

**November 2001**

© Copyright 2001 Hewlett-Packard Company

## **Notice**

© Copyright Hewlett-Packard Company 2001. All Rights Reserved.  
Reproduction, adaptation, or translation without prior written permission is prohibited, except as allowed under the copyright laws.

The information contained in this document is subject to change without notice.

Hewlett-Packard makes no warranty of any kind with regard to this material, including, but not limited to, the implied warranties of merchantability and fitness for a particular purpose. Hewlett-Packard shall not be liable for errors contained herein or for incidental or consequential damages in connection with the furnishing, performance or use of this material.

# Table Of Contents

Abstract .....	5
Introduction.....	5
Processor Sets Features Overview.....	5
Benefits of Processor Sets .....	7
Scheduling Allocation Domain .....	8
Configuring Processor Sets.....	8
Assigning Applications to Processor Sets.....	10
Ownership and Access Permissions Model.....	10
Attributes for Better Manageability .....	12
The Default Pset.....	14
System Daemon Processes.....	15
User Interfaces .....	15
Creating a New Processor Set .....	16
Assigning a Processor to Another Processor Set .....	17
Binding Applications to a Processor Set.....	18
Destroying a Processor Set.....	19
Querying and Changing Processor Set Attributes.....	19
Querying System and Processor Set Configuration.....	20
Maintaining Processor Set Configuration Across Reboots.....	22
Integration with PRM.....	23
Integration with iCOD .....	23
Integration with vPars.....	24
Use Models and Examples.....	24
Server Consolidation .....	24
Processor Isolation for Real Time Applications.....	25
Resource Partitioning among Users and Departments .....	25
Job Processing in Batch Mode .....	25
For More Information.....	26



---

## Abstract

This white paper discusses the HP-UX Processor Sets (Psets) feature in the HP-UX 11i release. The paper explains in detail the processor sets functionality and how systems can be configured with processor sets to effectively manage system processor resources. The paper also includes selected usage models and examples, as well as the benefits of using processor sets on HP-UX 11i systems.

---

## Introduction

HP-UX Processor Sets provide a flexible, lightweight mechanism for managing system processor resources among applications and users. A processor set consists of several processors grouped together for the exclusive access by applications assigned to that set. A processor set defines a *Scheduling Allocation Domain*. The operating system scheduler restricts applications to execute only on the processors in their assigned processor sets. Applications assigned to different processor sets do not contend with one another for processor resources. This isolation helps implement server consolidation on large systems.

HP-UX Processor Sets support dynamic runtime reconfiguration of processor sets and dynamic reassignment of workload among processor sets by users with the necessary privileges.

HP-UX Processor Sets are integrated with Hewlett-Packard's Process Resource Manager (PRM) for effective system resource management among multiple workloads, users, and departments within an enterprise.

HP-UX Processor Sets implementation is completely hardware independent, and can be used on any HP-UX 11i multi-processor system. HP-UX Processor Sets are available as an optional software product for HP-UX 11i systems.

---

## Processor Sets Features Overview

The key features of HP-UX Processor Sets include the following:

### Scheduling Allocation Domain

A processor set defines a Scheduling Allocation Domain for the applications in that processor set. An application is restricted to execute only on processors in its assigned processor set.

## **Dynamic Configuration**

Privileged users can dynamically set up and change the system processor set configuration. Users can create a new processor set, destroy an existing processor set, or reassign a processor from its current processor set to another. (Every enabled processor in the system is assigned to a processor set.)

## **Dynamic Application Binding**

Each running application in the system is bound to some processor set, and this processor set defines which processors the application may execute on. Privileged users can dynamically move (migrate) an application from its current processor set to another processor set in the system as needed. An application may also have binding to a specific processor in its processor set.

## **Ownership and Access Permissions**

HP-UX Processor Sets define a very flexible and powerful ownership and access permissions model, providing privileged users precise control in managing the system processor set configuration and workload distribution among processor sets. Every processor set has an owner along with a set of access permissions to grant or restrict access to other users in the system.

## **Attributes for Better Manageability**

HP-UX Processor Sets define a set of attributes that control system behavior for various operations on processor sets. One attribute, for example, lets a privileged user define whether to allow a user to destroy a processor set if the processor set is in use at the time the request is issued. Privileged users can dynamically change attribute values for a processor set as needed.

## **System Default Pset**

The HP-UX system is automatically configured with one System Default Processor Set (Default Pset) when started. The Default Pset has at least one processor, and its access permissions are set to provide access for all users in the system. The Default Pset is always available to all users.

## **Managing System Daemon Processes**

HP-UX has a set of kernel daemon processes to perform system-level activities. The HP-UX Processor Sets implementation does not restrict these daemon processes to any user-defined processor set configuration; rather, they are allowed to execute on any processor in the system irrespective of the user-defined processor set configuration.

## **Integration with PRM**

The HP-UX Process Resource Manager (PRM) software product provides a powerful tool for managing system resources among applications and users. PRM is now integrated with processor sets, allowing administrators to allocate processor resources to applications and users (through PRM groups) as a number of whole processors.

### **Member of Partitioning Continuum**

The HP-UX Partitioning Continuum provides a spectrum of tools and mechanisms for users to partition, allot, and manage system resources for their workloads, based on the degree of flexibility or isolation needed. HP-UX Processor Sets provide maximum flexibility in managing processor resource isolation among system workload.

---

## **Benefits of Processor Sets**

HP-UX Processor Sets add flexibility and power to the management of processor resources in server systems. The primary benefits of HP-UX Processor Sets are as follows:

### **Server Consolidation**

A dedicated set of processors can be assigned to a set of applications to take advantage of locality, and to prevent interference between applications. Processor resource partitioning can be implemented among different departments or user groups in an organization.

### **Integration with HP Process Resource Manager (PRM)**

PRM adds memory partitioning, automates process movement, maintains configuration across reboots, and provides GUI for configuration.

### **Dedicated Processor Resources for Batch Processing Jobs**

Each job in a processor set can be configured with the resources needed for its execution, thus facilitating better performance and workload management.

### **Service Special Needs**

Processors can be isolated for real-time applications. Applications that manage their own use of processor resources can now have whole processors.

### **Hardware and Platform Independent**

HP-UX 11i customers can use processor sets on all existing multi-processor systems.

### **Member of HP Partitioning Continuum**

HP-UX processor sets are seamlessly integrated with other components of the HP Partitioning Continuum (like nPartitions and Virtual Partitions).

---

## Scheduling Allocation Domain

A Scheduling Allocation Domain for an application defines which processors the application may execute on in the system. Without processor sets, an application has access to all processors in the system. Depending on the system workload scheduling policies and priorities, the operating system scheduler may execute the application on any processor in the system. We cannot isolate processor(s) for a given application. The entire system is the Scheduling Allocation Domain for an application. As a result, all applications compete with one another for processor cycles based on their scheduling policies and priorities. *(Note: If an application chooses explicit processor affinity, then the scheduler executes the application only on the processor chosen by the application).*

HP-UX Processor Sets provide privileged users the ability to define and manage a Scheduling Allocation Domain for an application. By grouping processors together for the exclusive use of specific applications, HP-UX Processor Sets define individual Scheduling Allocation Domains, thus preventing applications assigned to different processor sets from contending with one another for processor resources.

HP-UX Processor Sets support multiple instances of the POSIX Real Time Scheduler — one for every processor set. (Each POSIX Real Time Scheduler need to run on a global basis within its particular Scheduling Allocation Domain.) Applications using the POSIX Real Time scheduling policies in different processor sets do not contend with one another.

The HP-UX operating system load balancer works only *within* a processor set. The system does not implicitly perform any load balancing across processor sets. The user is responsible for ensuring the proper processor set configuration and workload distribution among the processor sets.

---

## Configuring Processor Sets

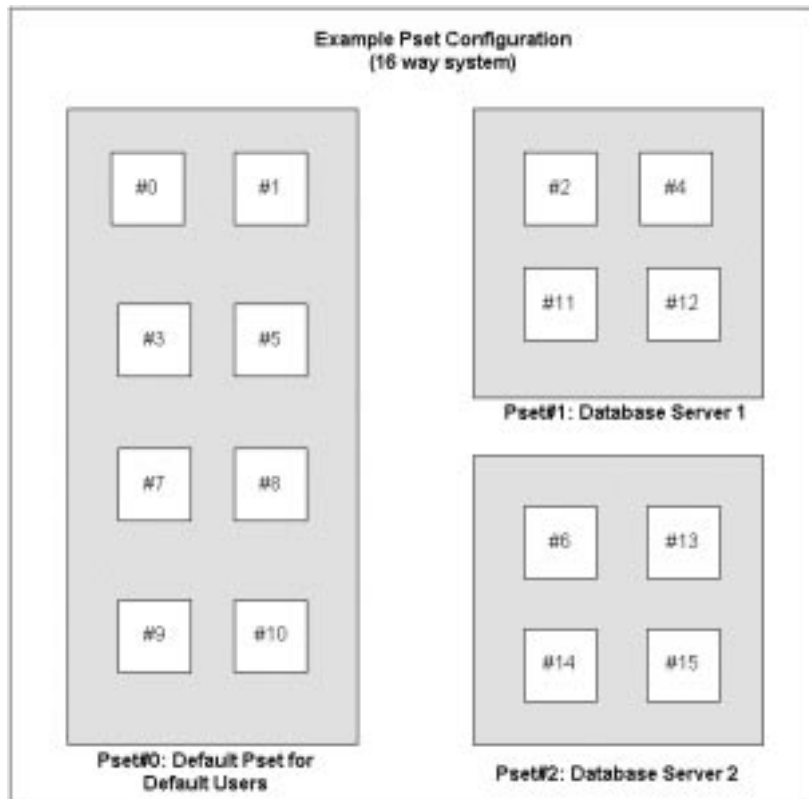
When booted, the HP-UX system starts with one user-visible processor set — the Default Pset — to which all enabled processors in the system are assigned. Users can then dynamically create additional new processor sets. A newly created processor set has no processors assigned to it. Every processor set is assigned a unique identifier (**psetid**) at creation, and this identifier can be used to perform further operations on the processor set. *(Note: Users do not have explicit control over which **psetid** is assigned to a newly created processor set).*

All enabled processors in the system have a processor set assignment, but a processor can only be assigned to just one user-defined processor set at a time. However, with the

appropriate access permissions, users can dynamically reassign any processor (except Processor 0) to another processor set, thereby removing the processor from its current set.

When all processors are removed from a processor set, the processor set becomes empty but still exists. Processors can be reassigned any time later to use the processor set again. However, an empty processor set cannot have any applications assigned to it.

Users can dynamically destroy a processor set when it is no longer needed. When a non-empty processor set (one that still has processors assigned to it) is destroyed, all its processors are reassigned to the Default Pset. Also, all applications bound to the non-empty processor set being destroyed are moved to the Default Pset.



**Figure 1: Example Processor Set Configuration**

The *Figure 1: Example Processor Set Configuration* shows an example Processor Set configuration on a 16-way system. The system is configured into three processor sets: one 8-processor processor set for all users, and two 4-processor processor sets for two different instances of database servers. The processors are randomly assigned to processor sets in this example.

---

## Assigning Applications to Processor Sets

Every process is bound to a processor set in the system. In a multi-threaded process, every thread has a processor set binding, but not all threads in the process need to be bound to the same processor set; an application may choose to bind some of its threads to different processor sets. As a result, a thread may have a different processor set binding than its process.

The processor set binding inheritance follows the scheduling inheritance rules. A newly created process and its first thread inherit their processor set binding from the parent process. A newly created thread in a multi-threaded process inherits its processor set binding from its creator thread. However, users can explicitly change the processor set binding of a process to another processor set in the system. When a process is migrated to another processor set, all its threads are also migrated to the same processor set.

Processor set binding inheritance, along with the ability to dynamically change processor set binding, enables excellent workload resource management capability. If an application needs exclusive processor resources, for instance, the application can be migrated to its own processor set with necessary processor resources. Users can also be restricted to specific processor sets. When a user logs into the system, migrate the first shell process for the user to the desired processor set, and from then on, all programs that the user executes will execute in the same processor set based on the inheritance rules.

A process or thread can also have processor binding that allows the operating system scheduler to ensure that the thread is always scheduled on the same processor. Note that the processor set binding of an application has higher precedence than the processor binding. Therefore, a process or thread cannot be bound to a processor outside its current processor set. Also, if a process or a thread with processor binding is migrated to another processor set, the processor binding for the process or thread is reassigned to a processor in the new processor set.

---

## Ownership and Access Permissions Model

HP-UX Processor Sets configuration and management operations are privileged operations. The Ownership and Access Permissions model for HP-UX Processor Sets provide necessary control and flexibility to meet the needs of a range of resource management use models. The key features of this Ownership and Access Permissions model are as follows:

- Superuser can perform any processor set operation.
- A new privilege group, PRIV\_PSET, is defined for processor set operations. (*Note: Please refer to the **setprivgrp(2)** manpage for more information on privilege groups in HP-UX.*) Users with PRIV\_PSET privilege can perform any processor set operation. This allows someone to acquire “restricted” superuser permissions for processor set management.
- Every processor set is assigned a user-owner, who may or may not have superuser or PRIV\_PSET privileges. The processor set creator becomes the default owner. Ownership is an attribute of a processor set; it may be changed to any other user with appropriate permissions. Ownership is identified by the user id (UID) of the owner.
- Every processor set is assigned a user group and is identified by the group id (GID) of the owner.
- All operations (except the topology query operations) on a processor set require some type of access permission.
- Ownership is used to define an access permissions model similar to the UNIX file system access permissions. Any user is in one of the Owner, Group, or Others categories, each with READ, WRITE and EXEC permissions for the user. Access permissions are an attribute of a processor set; they may be changed to grant or reject permission to a user for an operation on a processor set.

**Table 1: Ownership and Access Permissions Model for Processor Sets**

Pset Operation	Superuser	PRIV_PSET User	Users with Pset READ, WRITE, EXEC Access Permissions			Any Other User
			Source Pset	Target Pset	Permissions	
Create a Pset	Yes	Yes	N/A (Pset is not created as yet)			Yes
Destroy a Pset	Yes	Yes	Yes with WRITE permission			No
Reassign a processor to another Pset	Yes	Yes	Source Pset	Target Pset	Permissions	No
			Default Pset	Non Default Pset	N/A (Non-root and non-PRIV_PSET users cannot have WRITE permission to Default Pset)	
			Non Default Pset	Non Default Pset	Yes with WRITE permissions in both Psets	

			Non Default Pset	Default Pset	Yes with WRITE permission in Source Pset	
Bind a thread or a process to a Pset	Yes	Yes	Yes with EXEC permission if caller has same effective user-id (EUID) as that of the target process.			No
Change Pset attribute values (except Owner, Group, and Access permissions)	Yes	Yes	Yes with WRITE permission			No
Query Pset attribute values	Yes	Yes	Yes with READ permission			No
Change Owner, Group, or Access Permission	Yes	Yes	Only the Pset Owner			No
Query Pset configuration	Yes	Yes	Yes			Yes

---

## Attributes for Better Manageability

HP-UX Processor Sets define a set of attributes that allow users to control system behavior for various operations on processor sets. A processor set has default attribute values when started. Users may explicitly change the value of an attribute as needed. HP-UX processor sets support the following attributes and attribute values:

### **PSET\_ATTR\_OWNIID**

The UID (user id) of the processor set owner. At processor set creation time, the effective user ID of the creator is assigned as the owner.

### **PSET\_ATTR\_GRPID**

The GID (group id) of the processor set owner. The creator's effective group ID is assigned at processor set creation time.

### **PSET\_ATTR\_PERM**

The access permission bit mask for the processor set. These permissions, listed below, are defined similar to file access permissions:

- READ** User can query the processor set attributes.
- EXEC** User can bind its process or thread to the processor set.

**WRITE** User can modify the processor set attributes and configuration, destroy the processor set.

The following bit values are used for the access bit mask:

<b>PSET_OWNER_READ</b>	The processor set owner has READ access.
<b>PSET_OWNER_WRITE</b>	The processor set owner has WRITE access.
<b>PSET_OWNER_EXEC</b>	The processor set owner has EXEC access.
<b>PSET_GROUP_READ</b>	All users in the processor set owner's group have READ access.
<b>PSET_GROUP_WRITE</b>	All users in processor set owners group have WRITE access.
<b>PSET_GROUP_EXEC</b>	All users in processor set owner's group have EXEC access.
<b>PSET_OTHER_READ</b>	All others have READ access.
<b>PSET_OTHER_WRITE</b>	All others have WRITE access.
<b>PSET_OTHER_EXEC</b>	All others have EXEC access.

A newly created processor set provides READ and EXEC access for everyone, but WRITE access only for the processor set owner.

#### **PSET\_ATTR\_NONEMPTY**

Indicates the behavior when a request is made to destroy a non-empty processor set. A processor set is non-empty when it has at least one processor assigned to it. The following values are supported:

<b>PSET_ATTRVAL_DFLTPSET</b>	Reassign all processors in the processor set to the Default Pset. Migrate all threads and processes in the processor set to the Default Pset. This is the default value for this attribute.
<b>PSET_ATTRVAL_FAIL</b>	Fail the request with busy error if there are threads or processes bound to the processor set or if there are processors assigned to the processor set.
<b>PSET_ATTRVAL_FAILBUSY</b>	Fail the request with busy error only if there are active threads or processes bound to the processor set. Otherwise, perform the operation by reassigning the processors in the processor set to the Default Pset.

## PSET\_ATTR\_EMPTY

Indicates the behavior when a request to bind a process or a thread to an empty processor set is made. An empty processor set has no processors assigned to it. Currently, only the following value is supported:

**PSET\_ATTRVAL\_FAIL** Reject the request. This is the default value for this attribute.

## PSET\_ATTR\_LASTSPU

Indicates the behavior when a request to remove the last processor from a processor set is made. A processor may be removed from a processor set to be reassigned to another processor set, or to be disabled from processing any workload in the system. The following values are supported:

**PSET\_ATTRVAL\_FAIL** Fail the request if there are active threads and processes assigned to the processor set.

**PSET\_ATTRVAL\_DFLTPSET** Remove the processor from the processor set, and migrate all threads and processes to the Default Pset. This is the default value for this attribute.

---

## The Default Pset

HP-UX systems start with one user-visible processor set when booted: the Default Pset. During system initialization, all enabled processors in the system are assigned to the Default Pset, which is assigned the unique **psetid** of PS\_DEFAULT. The Default Pset has special significance in the HP-UX Processor Sets model. Its key features are as follows:

- The Default Pset can never be destroyed.
- Processor 0 is always assigned to the Default Pset, and cannot be reassigned to any other user-created processor set.
- The superuser is the Default Pset owner and can never be changed.
- The superuser or a PRIV\_PSET privileged user may reassign any processor (except Processor 0) out of the Default Pset to other user-created processor sets .
- The Default Pset is always accessible to all applications and users in the system. Its access permissions cannot be changed to restrict access to any users.
- The Default Pset acts as the final home for the deserted. When a processor set is destroyed, all applications and processors in that processor set are reassigned to the

Default Pset. When the last processor from a processor set is reassigned to another processor set, all applications in the now-to-be-empty processor set are migrated to the Default Pset.

---

## System Daemon Processes

The HP-UX operating system provides a set of kernel-private daemon processes to perform system-level activities and services. Users may configure processor sets in such a way that only a minimum number of processors are left in the Default Pset. If kernel daemon processes were to be restricted to the Default Pset, they may get starved. Therefore, since system responsiveness and the progress of user applications depend on these daemon processes running regularly as needed, HP-UX Processor Sets do not restrict the kernel daemons to processor set boundaries. Instead, HP-UX has introduced the concept of the **Kernel Pset** for these daemon processes. (*Note: These daemon processes do not include the user space daemon processes.*) The key features of the Kernel Pset are as follows:

- All processors in the system are by default assigned to the Kernel Pset. Kernel daemon processes and threads may run on any processors in the Kernel Pset.
- Processors in the Default Pset are always available to the Kernel Pset.

Since user daemon processes are assigned by default to the Default Pset, the administrator should assign sufficient processor resources to the Default Pset to ensure that those user daemons critical to system responsiveness do not get starved. Some examples of user-space daemons that can have impact on the overall system responsiveness and throughput are as follows:

- If the system is configured with remote file systems using NFS (Network File System), the responsiveness of NFS daemons is critical to performance of applications that access these remote file systems.
- For server systems that have a large number of remote users who access the system through remote logins, the responsiveness of **inetd** and **rlogind** daemons is important to provide smooth and quick access to remote users.

Alternately, the administrator can create a new processor set dedicated for these critical user daemons and then allocate sufficient processor resources to ensure overall system progress.

---

## User Interfaces

To programmatically manage system processor set configuration and workload assignment, HP-UX Processor Sets support the following application programming interfaces (APIs) for applications:

**Table 2. Supported Pset APIs**

API	Description
pset_create(2)	Create a new processor set.
pset_destroy(2)	Destroy a processor set.
pset_assign(2)	Reassign a processor from one processor set to another.
pset_bind(2)	Rebind a process or a thread from one processor set to another.
pset_setattr(2)	Change processor set attribute values.
pset_getattr(2)	Query processor set attribute values.
pset_ctl(2)	Query processor set configuration.
pthread_pset_bind_np(3t)	Change the pset binding of a pthread in multi-threaded applications.
mpctl(2)	Apply processor affinity to applications within the application's processor set. Query system and processor set configuration.
pstat_getpset(2)	Query processor set information using the HP-UX pstat interface mechanism.

The **psrset(1m)** tool provides a command line interface to manage system processor set configuration and workload assignment on HP-UX systems. The **psrset** command provides the following functionality:

- Create a new processor set
- Assign processors to a processor set
- Destroy processor sets
- Migrate a running application to another processor set
- Start an application in a specific processor set
- Query processor set configuration, application's current processor set
- Query and change processor set attributes

The system monitoring and query commands like **top(1)**, **uptime(1)**, **ps(1)** and **mpsched(1)** are enhanced to provide options to display processor set specific information.

### **Creating a New Processor Set**

The **pset\_create()** system call creates a new, empty processor set with no processors. A unique identifier (**psetid**) is assigned to the new processor set and returned to the user.

```

int
pset_create(
    psetid_t *newpset /* OUT: Id of new pset */
);

```

Any user may create a new empty processor set object. This relaxation in processor set creation, when coupled with the access permissions for other processor set operations, provides for a very flexible processor set usage model. Consider this scenario: The system administrator configures a processor set with some processor resources and then makes some other department administrator owner of the processor set. Without the need for superuser or PRIV\_PSET privileges, this department administrator can then further subdivide these processors into multiple processor sets and allocate them to users in his or her department without relying on the system administrator.

HP-UX enforces a per-user and system-wide implementation-dependent limit on the number of processor sets that can be active at any time. The first release of HP-UX Processor Sets has an internal fixed limit based on the number of processors in the system. There is no tunable to control the maximum number of processor sets in the first release.

Users can create a processor set from the command line as shown below. (Note: The user does not have explicit control over which psetid is assigned to the new processor set.)

```

# psrset -c
successfully created pset 1

```

### **Assigning a Processor to Another Processor Set**

The **pset\_assign()** system call allows users to reassign a processor from its current processor set to another processor set.

```

int
pset_assign(
    psetid_t new_pset, /* IN : new pset for the processor */
    spu_t    spu,      /* IN : processor to be reassigned */
    psetid_t *old_pset /* OUT: current pset for processor */
);

```

If `spu` is the last processor in its current processor set, then the `PSET_ATTR_LASTSPU` attribute for the current processor set determines its behavior. A processor set can have zero, one, or more processors in the system (with the restriction that processor 0 can be assigned only to the Default Pset). Users should assess the need of applications in determining the size of a processor set. If a processor set is under-allocated, the applications in that processor set may starve out due to contention on processors. If a processor set is over-allocated, there will be idle cycles in the processor set, while applications in other processor sets in the system may have contention on processors.

Users can reassign a list of processors to another processor set using **psrset** as follows:

```
# psrset -a 1 2 4 6
successfully assigned processor 2 to pset 1
successfully assigned processor 4 to pset 1
successfully assigned processor 6 to pset 1
```

Users can create a pset and assign processors using **psrset** as follows:

```
# psrset -c 2 4 6
successfully created pset 2
successfully assigned processor 2 to pset 2
successfully assigned processor 4 to pset 2
successfully assigned processor 6 to pset 2
```

## **Binding Applications to a Processor Set**

The **pset\_bind()** system call allows users to rebind a process, a thread, all processes owned by some user, or all processes in a process group from its current processor set to another processor set. When binding for a process is changed, binding for all its threads is also changed to the new processor set.

```
int
pset_bind (
    psetid_t pset,      /* IN: new pset for process/thread */
    idtype_t idtype,   /* IN: P_PID,P_LWPID,P_UID,P_PGRP */
    id_t id,          /* IN: pid, lwpid, uid or pgrp */
    psetid_t* old_pset /* OUT: current pset */
);
```

If `pset` is empty, then the `PSET_ATTR_EMPTY` attribute for the processor set defines the behavior. If `idtype` is `P_UID`, the processor set binding of all processes with effective user IDs the same as `id` is changed. If `idtype` is `P_PGRP`, the processor set binding of all processes in the process group specified by `id` is changed.

Users can rebind running processes to another processor set by specifying the process ID (PID), the effective user ID (UID), or the process group ID (PGRP) values:

```
# psrset -b 1 232
successfully bound pid 232 to pset 1

# psrset -U 1 21
successfully bound processes owned by uid 21 to pset 1

# psrset -g 1 20
successfully bound processes of pgid 20 to pset 1
```

Users can start an application in another processor set using **psrset**. The following example will run the `ps -ef` command in pset 2:

```
# psrset -e 2 ps -ef
```

The **mpctl()** system call allows an application to bind to a specific processor for better locality performance. However, the processor sets restrict the processor affinity to a processor within the application's processor set. Users can bind a running process to a specific processor using **mpsched** as follows:

```
# mpsched -c 1 -p 271
Pid 271: bound to processor 1
```

Multi-threaded applications based on the POSIX PTHREADS library can change the processor set binding of a pthread using the **pthread\_pset\_bind\_np()** library interface.

```
int
pthread_pset_bind_np(
    psetid_t      *answer, /* OUT: thread's current pset*/
    psetid_t      pset,    /* IN : new pset for thread */
    pthread_t pthreadid /* IN : target pthread */
);
```

### **Destroying a Processor Set**

The **pset\_destroy()** system call allows users to destroy a processor set.

```
int
pset_destroy(
    psetid_t pset /* IN: pset to be destroyed */
);
```

If `pset` is non-empty or has bound applications, then the `PSET_ATTR_NONEMPTY` attribute's value for the processor set determines its behavior. While the Default Pset cannot be destroyed, users can destroy other processor sets using **psrset** as follows:

```
# psrset -d 1 2
successfully destroyed pset 1
successfully destroyed pset 2
```

The following command will try to destroy all processor sets except the Default Pset if the caller has appropriate permissions and the `PSET_ATTR_NONEMPTY` attribute for the processor set allows the operation.

```
# psrset -d all
```

### **Querying and Changing Processor Set Attributes**

The **pset\_getattr()** system call allows users to query attribute values of a processor set.

```
int
pset_getattr (
    psetid_t      pset, /* IN : target pset */
    pset_attrtype_t type, /* IN : which attribute */
    pset_attrval_t* val /* OUT: current value of attr */
);
```

The **pset\_setattr()** system call allows users to change the value of an attribute of a processor set.

```
int
pset_setattr (
    psetid_t      pset, /* target pset */
    pset_attrtype_t type, /* which attribute */
    pset_attrval_t val /* current value of attr */
);
```

```

    pset_attrtype_t type, /* which attribute */
    pset_attrval_t   val/* new value for attribute */
);

```

(Please see the section, *Attributes for Better Manageability*, for the list of supported attributes and possible attribute values.)

Users can query and change attribute values using **psrset**. In the following example, the user first queries the current values of attributes for processor set 1, and then changes the processor set ownership, access permissions, and the value of the PSET\_ATTR\_NONEMPTY attribute.

```

# psrset -i 1
PSET ID      1
SPU_LIST     2
OWNID        0
GRPID        3
PERM         755
NONEMPTY     DFLTPSET
EMPTY        FAIL
LASTSPU      DFLTPSET

# psrset -t 1 NONEMPTY=FAILBUSY
# psrset -t 1 OWNID=100
# psrset -t 1 GRPID=21
# psrset -t 1 PERM=700

# psrset -i 1
PSET ID      1
SPU_LIST     2
OWNID        100
GRPID        21
PERM         700
NONEMPTY     FAILBUSY
EMPTY        FAIL
LASTSPU      DFLTPSET

```

### **Querying System and Processor Set Configuration**

The **pset\_ctl()** system call allows users to query system processor set configuration. Any user may query the system processor set configuration.

```

int
pset_ctl (
    pset_request_t request, /* query request */
    psetid_t      pset,    /* target pset */
    id_t          id       /* request dependent object */
);

```

**Table 3. Requests Supported by the pset\_ctl() Interface.**

Request	Description
PSET_GETNUMPSETS	Return the current number of processor sets in the

	system.
PSET_GETFIRSTPSET	Return <b>psetid</b> of the first processor set in the system.
PSET_GETNEXPSET	Return <b>psetid</b> of the next processor set in the system after <code>pset</code> .
PSET_GETCURRENTPSET	Return <b>psetid</b> of the current processor set for the calling thread.
PSET_GETNUMSPUS	Return the number of processors assigned to the processor set <code>pset</code> .
PSET_GETFIRSTSPU	Return the ID of the first processor in the processor set <code>pset</code> .
PSET_GETNEXTSPU	Return the ID of next processor in the processor set <code>pset</code> after the processor specified in <code>id</code> .
PSET_SPUTOPSET	Return the ID of the processor set assigned for the processor specified in <code>id</code> .

The **mpctl()** system call interface provides query operations for system-level and processor set level configuration:

```
int
mpctl(
    mpc_request_t request, /* IN: request type      */
    spu_t        spu,     /* IN: spu id, request */
    pid_t        id,      /* IN: request dependent */
);
```

The **mpctl()** interface provides two sets of query requests:

1. An application can use the **mpctl()** interface to query which processors are available in its current processor set, so that it can bind and scale for optimal performance. Only the processors in the application's processor set are now available to the application. The **mpctl()** query requests return the processor information in the calling thread's processor set.
2. The tools and monitor programs can use the **mpctl()** interface to query the system level configuration information regardless of the system's processor set configuration.

**Table 4. Query Requests Supported by the mpctl() Interface.**

Request	Description
MPC_GETNUMSPUS	Return number of processors in the calling thread's processor set.
MPC_GETFIRSTSPU	Return ID of the first processor in the calling thread's processor set.
MPC_GETNEXTSPU	Return ID of the next processor in the calling thread's processor set after the specified processor.
MPC_GETCURRENTSPU	Return ID of the current processor for the calling

	thread.
MPC_GETNUMSPUS_SYS	Return number of enabled processors in the system.
MPC_GETFIRSTSPU_SYS	Return ID of the first enabled processor in the system.
MPC_GETNEXTSPU_SYS	Return ID of the next enabled processor in the system after the specified processor.

The **pstat\_getpset()** interface allows applications to query the processor sets specific information.

```
int
pstat_getpset(
    struct pst_pset *buf, /* OUT: buffer for pset information */
    size_t          elemsize, /* IN: buf size for each entry */
    size_t          elemcnt, /* IN: how many psets we query */
    int            index /* IN: which pset to start with */
);
```

This interface returns the following information in the buffer for every processor set being queried:

- Psetid of the processor set
- Number of processors assigned to the processor set
- Number of processes bound to the processor set
- Number of threads bound to the processor set
- Owner and Group of the processor set
- Access permissions bitmask for the processor set
- Load average values for the processor pset

If invoked without any arguments, the **psrset** command returns the current system processor set configuration for all processor sets. It returns information about a specific processor set with the **-i** option. The **mpsched** command returns the system configuration without any processor set information. The **top** command allows users to display activity in a specific processor set, or to display processor set assignments for processors and processes.

### **Maintaining Processor Set Configuration Across Reboots**

The processor set configuration on HP-UX systems is not persistent across reboots. When a system is shut down, the configuration is lost, and by default, the system starts with one processor set – the Default Pset – when rebooted. There are two possible ways to retain the same processor set configuration across reboots:

- **Use PRM.** The HP PRM maintains the configuration in a database. When PRM is started after reboot, it can restore the configuration saved in the database.
- **Use psrset.** The **psrset** command provides a command line interface for processor set configuration, which can be programmed as a shell script for the desired configuration. The administrator can set up the invocation of the processor set configuration script from the system boot time script, so that the system is already set up with the desired processor set configuration when the system is ready for use.

---

## Integration with PRM

HP-UX Process Resource Manager (PRM) is a software product for managing system resources (processor, memory, disk I/O bandwidth) among applications and users. The HP-UX PRM administrator assigns applications and users to PRM groups, and sets up each PRM group with a portion of system resources. PRM assigns processor resource entitlements (guaranteed minimums) to each PRM group as a number of shares, and the HP-UX scheduler enforces the shares allocation among PRM groups. The allocation of shares ensures a certain percentage of processor cycles from every processor to PRM groups.

With processor sets integration, PRM can now allocate processor resources to PRM groups as entire processors. The PRM administrator may specify that a PRM group is now a Pset PRM group. The Pset PRM group, instead of receiving shares of available processor cycles, is now assigned a specified number of processors. PRM continues to support shares-based PRM groups within the Default Pset.

PRM-based system resource management provides the following benefits:

- PRM supports memory resource allocation to PRM groups. When coupled with Psets, the administrator can isolate processors and memory in a PRM group for applications and users.
- PRM provides a GUI for configuration of psets and workload assignment.
- PRM can preserve the same configuration across reboots.
- PRM allows second level of resource management using shares within Default Pset.
- PRM automatically moves workloads (applications and users) into the appropriate PRM groups.

---

## Integration with iCOD

The HP-UX iCOD (Instant Capacity On Demand) product allows dynamic deactivation and reactivation of processors in the system, based on user needs. The system may be

configured with maximum processors, but users activate only the processors needed to meet their needs. A deactivated processor remains in the system (ready to be activated at any time without requiring system reboot), but does not process any workload. The processor sets will integrate with iCOD as follows:

- A processor is removed from its current processor set when deactivated. A deactivated processor does not belong to any processor set.
- When reactivated, the processor is added to the Default Pset. The user has to explicitly assign it to other processor set if needed.
- Just as in reassignment, deactivation of the last processor in a processor set is governed by the value of the PSET\_ATTR\_LASTSPU attribute for the processor set. If the attribute value is defined to fail the request, iCOD may not disable the last processor in the processor set.

The complete support for iCOD with processor sets will be provided with release of iCOD version 5.0 or later.

---

## Integration with vPars

The HP-UX Virtual Partitions (vPars) software product allows users to configure their systems in multiple logical partitions where each partition is assigned its own set of processor, memory, and I/O resources. Each virtual partition runs a separate instance of HP-UX operating system. Users can dynamically migrate processors from one virtual partition to another without requiring reboot of any virtual partition. The HP-UX Processor Sets are integrated with vPARs, and support dynamic processor migration as follows:

- When a processor is removed from its current vPAR instance, it is also removed from its current processor set.
- When a processor is added to a vPar instance, the processor is added to the Default Pset in that vPar instance.

---

## Use Models and Examples

The following realistic use models and examples illustrate how processor sets make a positive difference for customers.

### **Server Consolidation**

Many customers are accustomed to isolating their critical applications on separate servers in order to minimize interference among applications, and to achieve the required performance.

Also, many applications are designed to scale to all available resources in the system with the assumption that the entire system is available to them.

As servers with enough power to host many applications become available, the customers are now looking to consolidate their applications into larger servers, although they still need resource partitioning among applications to minimize interference and to guarantee necessary resource allocation to each application. Processor sets provide an excellent lightweight and flexible mechanism to partition a system's processor resources among the applications residing on the same server. For example, different departments in an organization who have been running their own separate database servers, may now have their database servers hosted on a larger HP-UX server. When used with HP-UX PRM, each instance of the database servers can be assigned a subset of processors as well as memory in the system.

### **Processor Isolation for Real Time Applications**

Real time applications expect somewhat deterministic responses or workload completion times, and hence want zero-to-minimal interference from other running applications in the system. In the past, there was no way to achieve this, so customers set up separate systems for real-time applications. With processor sets, however, customers can now use a single system to achieve the same objectives. Within the system, they can configure a processor set with the required number of processors and bind only the real-time application to that processor set.

### **Resource Partitioning among Users and Departments**

Departments in an organization may be running their own email servers, as well as timeshare servers for their users. Processor sets allow these servers to be consolidated into larger systems.

Different departments may also have varying resource needs and budget constraints. With processor sets, processor resources can be assigned and managed among users of different departments based on their budget allocations.

### **Job Processing in Batch Mode**

When processing jobs in a batch mode, the jobs are scheduled one after another, and essentially no resource allocation is made on a per-job basis. In contrast, the dynamic configuration feature of processor sets allows the system user to configure a processor set for a given job based on the processor resource requirements for the job. The user can then execute the job in the processor set and, when the job is finished, destroy the set. In short, processor sets provide for a more deterministic job completion, and allow concurrent scheduling of multiple jobs based on resource requirements and system size.

---

## For More Information

Please refer to the following web sites for more information on processor sets, resource management, and other features available in HP-UX:

- HP-UX Operating System  
<http://www.hp.com/go/hpux>
- HP-UX Processor Sets  
[http://www.hp.com/products1/unix/operating/hpux11i/hpux11i\\_proc\\_sets.html](http://www.hp.com/products1/unix/operating/hpux11i/hpux11i_proc_sets.html)
- Download Processor Sets for HP-UX 11i systems  
<http://ww.software.hp.com> (Click Enhancement Releases)
- HP-UX Process Resource Manager (PRM)  
<http://ww.hp.com/go/prm>
- HP-UX Workload Manager (WLM)  
<http://www.hp.com/go/wlm>
- HP Partitioning Continuum, HP Virtual Partitions  
<http://www.hp.com/go/servicecontrol>
- HP-UX Instant Capacity on Demand (iCOD)  
<http://www.hp.com/go/icod>
- HP System Partitions Guide: Administration for nPartitions  
<http://docs.hp.com> (Click on HP-UX 11i Operating Environments)