The illumos

Memory and Thread Placement Optimization Developer's Guide

Contents

Contents i

List of Figures ii

List of Tables ii

1 Locality Group APIs 1
- 1.1 Locality Groups Overview . . . 1
- 1.2 Verifying the Interface Version . . . 3
- 1.3 Initializing the Locality Group Interface . . . 3
  - Using `lgrp_init` . . . 4
  - Using `lgrp_fini` . . . 4
- 1.4 Locality Group Hierarchy . . . 4
  - Using `lgrp_cookie_stale` . . . 5
  - Using `lgrp_view` . . . 5
  - Using `lgrp_nlgrps` . . . 5
  - Using `lgrp_root` . . . 5
  - Using `lgrp_parents` . . . 6
  - Using `lgrp_children` . . . 6
- 1.5 Locality Group Contents . . . 6
  - Using `lgrp_resources` . . . 6
  - Using `lgrp_cpus` . . . 7
  - Using `lgrp_mem_size` . . . 7
- 1.6 Locality Group Characteristics . . . 8
  - Using `lgrp_latency_cookie` . . . 8
- 1.7 Locality Groups and Thread and Memory Placement . . . 9
  - Using `lgrp_home` . . . 9
  - Using `madvise` . . . 10
  - Using `madv.so.1` . . . 11
    - `madv.so.1` Usage Examples . . . 12
  - Using `meminfo` . . . 13
  - Locality Group Affinity . . . 15
    - Using `lgrp_affinity_get` . . . 15
    - Using `lgrp_affinity_set` . . . 16
- 1.8 Examples of API Usage . . . 16

2 MPO Observability Tools 25
- 2.1 The `pmadvise` utility . . . 25
- 2.2 The `plgrp` tool . . . 26

Specifying Lgroups . . . 27
Specifying Process and Thread Arguments . . . 27
2.3 The `lgrpinfo` Tool . . . 28
Options for the `lgrpinfo` Tool . . . 28
2.4 The `Solaris::lgrp` Module . . . 29

A Document License **33**
A.1 Public Documentation License (PDL), Version 1.01 . . . 33
A.2 Public Documentation License Notice . . . 37

List of Figures

1.1 Single Locality Group Schematic . . . 2
1.2 Multiple Locality Groups Schematic . . . 2

List of Tables

1 Typographic Conventions . . . iii
2 Shell Prompts . . . iv

Preface

The *Memory and Thread Placement Optimization Developer's Guide* provides information on locality groups and the technologies that are available to optimize the use of computing resources in the illumos operating system.

Who Should Use This Book

This book is intended for use by developers who are writing applications in an environment with multiple CPUs and a non-uniform memory architecture. The programming interfaces and tools that are described in this book give the developer control over the system's behavior and resource allocation.

Related Third-Party Web Site References

Third-party URLs are referenced in this document and provide additional, related information.

Note

Sun is not responsible for the availability of third-party web sites mentioned in this document. Sun does not endorse and is not responsible or liable for any content, advertising, products, or other materials that are available on or through such sites or resources. Sun will not be responsible or liable for any actual or alleged damage or loss caused or alleged to be caused by or in connection with use of or reliance on any such content, goods, or services that are available on or through such sites or resources.

Typographic Conventions

The following table describes the typographic conventions that are used in this book.

Table 1: Typographic Conventions

Typeface	Meaning	Example
`AaBbCc123`	The names of commands, files, and directories, and onscreen computer output	Edit your `.login` file. Use **ls** `-a` to list all files. `machine_name% you have mail.`

Table 1: (continued)

Typeface	Meaning	Example
AaBbCc123	What you type, contrasted with onscreen computer output	machine_name% **su** Password:
aabbcc123	Placeholder: replace with a real name or value	The command to remove a file is **rm** *filename*. Read Chapter 6 in the *User's Guide*.
AaBbCc123	Book titles, new terms, and terms to be emphasized	A *cache* is a copy that is stored locally. Do *not* save the file. **Note:** Some emphasized items appear bold online.

Shell Prompts in Command Examples

The following table shows the default UNIX® system prompt and superuser prompt for the C shell, Bourne shell, and Korn shell.

Table 2: Shell Prompts

Shell	Prompt
C shell	machine_name%
C shell for superuser	machine_name#
Bourne shell and Korn shell	$
Bourne shell and Korn shell for superuser	#

Chapter 1

Locality Group APIs

This chapter describes the APIs that applications use to interact with locality groups.

This chapter discusses the following topics:

- Section 1.1 describes the locality group abstraction.
- Section 1.2 describes the functions that give information about the interface.
- Section 1.3 describes function calls that initialize and shut down the portion of the interface that is used to traverse the locality group hierarchy and to discover the contents of a locality group.
- Section 1.4 describes function calls that navigate the locality group hierarchy and functions that get characteristics of the locality group hierarchy.
- Section 1.5 describes function calls that retrieve information about a locality group's contents.
- Section 1.6 describes function calls that retrieve information about a locality group's characteristics.
- Section 1.7 describes how to affect the locality group placement of a thread and its memory.
- Section 1.8 contains code that performs example tasks by using the APIs that are described in this chapter.

1.1 Locality Groups Overview

Shared memory multiprocessor computers contain multiple CPUs. Each CPU can access all of the memory in the machine. In some shared memory multiprocessors, the memory architecture enables each CPU to access some areas of memory more quickly than other areas.

When a machine with such a memory architecture runs the illumos software, providing information to the kernel about the shortest access times between a given CPU and a given area of memory can improve the system's performance. The locality group (lgroup) abstraction has been introduced to handle this information. The lgroup abstraction is part of the Memory Placement Optimization (MPO) feature.

An lgroup is a set of CPU–like and memory–like devices in which each CPU in the set can access any memory in that set within a bounded latency interval. The value of the latency interval represents the least common latency between all the CPUs and all the memory in that lgroup. The latency bound that defines

an lgroup does not restrict the maximum latency between members of that lgroup. The value of the latency bound is the shortest latency that is common to all possible CPU-memory pairs in the group.

Lgroups are hierarchical. The lgroup hierarchy is a Directed Acyclic Graph (DAG) and is similar to a tree, except that an lgroup might have more than one parent. The root lgroup contains all the resources in the system and can include child lgroups. Furthermore, the root lgroup can be characterized as having the highest latency value of all the lgroups in the system. All of its child lgroups will have lower latency values. The lgroups closer to the root have a higher latency while lgroups closer to leaves have lower latency.

A computer in which all the CPUs can access all the memory in the same amount of time can be represented with a single lgroup (see Figure 1.1). A computer in which some of the CPUs can access some areas of memory in a shorter time than other areas can be represented by using multiple lgroups (see Figure 1.2).

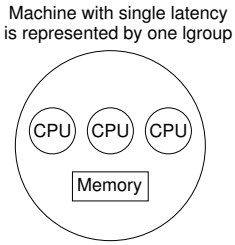

Figure 1.1: Single Locality Group Schematic

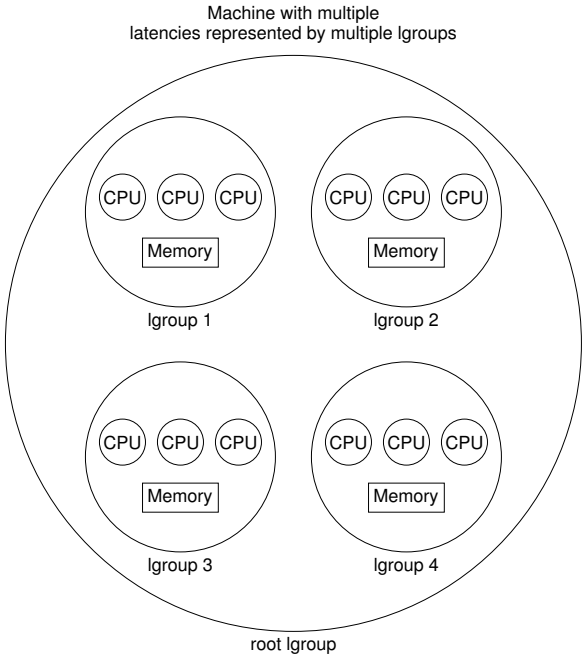

Figure 1.2: Multiple Locality Groups Schematic

The organization of the lgroup hierarchy simplifies the task of finding the nearest resources in the system. Each thread is assigned a home lgroup upon creation. The operating system attempts to allocate resources for the thread from the thread's home lgroup by default. For example, the illumos kernel attempts to

schedule a thread to run on the CPUs in the thread's home lgroup and allocate the thread's memory in the thread's home lgroup by default. If the desired resources are not available from the thread's home lgroup, the kernel can traverse the lgroup hierarchy to find the next nearest resources from parents of the home lgroup. If the desired resources are not available in the home lgroup's parents, the kernel continues to traverse the lgroup hierarchy to the successive ancestor lgroups of the home lgroup. The root lgroup is the ultimate ancestor of all other lgroups in a machine and contains all of the machine's resources.

The lgroup APIs export the lgroup abstraction for applications to use for observability and performance tuning. A new library, called `liblgrp`, contains the new APIs. Applications can use the APIs to perform the following tasks:

- Traverse the group hierarchy
- Discover the contents and characteristics of a given lgroup
- Affect the thread and memory placement on lgroups

1.2 Verifying the Interface Version

The lgrp_version(3LGRP) function must be used to verify the presence of a supported lgroup interface before using the lgroup API. The `lgrp_version` function has the following syntax:

```
#include <sys/lgrp_user.h>
int lgrp_version(const int version);
```

The `lgrp_version` function takes a version number for the lgroup interface as an argument and returns the lgroup interface version that the system supports. When the current implementation of the lgroup API supports the version number in the `version` argument, the `lgrp_version` function returns that version number. Otherwise, the `lgrp_version` function returns `LGRP_VER_NONE`.

Example 1.1: Example of `lgrp_version` Use

```
#include <sys/lgrp_user.h>
if (lgrp_version(LGRP_VER_CURRENT) != LGRP_VER_CURRENT) {
    fprintf(stderr, "Built with unsupported lgroup interface %d\n",
        LGRP_VER_CURRENT);
    exit (1);
}
```

1.3 Initializing the Locality Group Interface

Applications must call lgrp_init(3LGRP) in order to use the APIs for traversing the lgroup hierarchy and to discover the contents of the lgroup hierarchy. The call to `lgrp_init` gives the application a consistent snapshot of the lgroup hierarchy. The application developer can specify whether the snapshot contains only the resources that are available to the calling thread specifically or the resources that are available to the operating system in general. The `lgrp_init` function returns a cookie that is used for the following tasks:

- Navigating the lgroup hierarchy

- Determining the contents of an lgroup
- Determining whether the snapshot is current

Using `lgrp_init`

The `lgrp_init` function initializes the lgroup interface and takes a snapshot of the lgroup hierarchy.

```
#include <sys/lgrp_user.h>
lgrp_cookie_t lgrp_init(lgrp_view_t view);
```

When the `lgrp_init` function is called with `LGRP_VIEW_CALLER` as the view, the function returns a snapshot that contains only the resources that are available to the calling thread. When the `lgrp_init` function is called with `LGRP_VIEW_OS` as the view, the function returns a snapshot that contains the resources that are available to the operating system. When a thread successfully calls the `lgrp_init` function, the function returns a cookie that is used by any function that interacts with the lgroup hierarchy. When a thread no longer needs the cookie, call the `lgrp_fini` function with the cookie as the argument.

The lgroup hierarchy consists of a root lgroup that contains all of the machine's CPU and memory resources. The root lgroup might contain other locality groups bounded by smaller latencies.

The `lgrp_init` function can return two errors. When a view is invalid, the function returns EINVAL. When there is insufficient memory to allocate the snapshot of the lgroup hierarchy, the function returns ENOMEM.

Using `lgrp_fini`

The lgrp_fini(3LGRP) function ends the usage of a given cookie and frees the corresponding lgroup hierarchy snapshot.

```
#include <sys/lgrp_user.h>
int lgrp_fini(lgrp_cookie_t cookie);
```

The `lgrp_fini` function takes a cookie that represents an lgroup hierarchy snapshot created by a previous call to `lgrp_init`. The `lgrp_fini` function frees the memory that is allocated to that snapshot. After the call to `lgrp_fini`, the cookie is invalid. Do not use that cookie again.

When the cookie passed to the `lgrp_fini` function is invalid, `lgrp_fini` returns EINVAL.

1.4 Locality Group Hierarchy

The APIs that are described in this section enable the calling thread to navigate the lgroup hierarchy. The lgroup hierarchy is a directed acyclic graph that is similar to a tree, except that a node might have more than one parent. The root lgroup represents the whole machine and contains all of that machine's resources. The root lgroup is the lgroup with the highest latency value in the system. Each of the child lgroups contains a subset of the hardware that is in the root lgroup. Each child lgroup is bounded by a lower latency value. Locality groups that are closer to the root have more resources and a higher latency. Locality groups that are closer to the leaves have fewer resources and a lower latency. An lgroup can contain resources directly within its latency boundary. An lgroup can also contain leaf lgroups that contain their own sets of resources. The resources of leaf lgroups are available to the lgroup that encapsulates those leaf lgroups.

Using `lgrp_cookie_stale`

The lgrp_cookie_stale(3LGRP) function determines whether the snapshot of the lgroup hierarchy represented by the given cookie is current.

```
#include <sys/lgrp_user.h>
int lgrp_cookie_stale(lgrp_cookie_t cookie);
```

The cookie returned by the `lgrp_init` function can become stale due to several reasons that depend on the view that the snapshot represents. A cookie that is returned by calling the `lgrp_init` function with the view set to `LGRP_VIEW_OS` can become stale due to changes in the lgroup hierarchy such as dynamic reconfiguration or a change in a CPU's online status. A cookie that is returned by calling the `lgrp_init` function with the view set to `LGRP_VIEW_CALLER` can become stale due to changes in the calling thread's processor set or changes in the lgroup hierarchy. A stale cookie is refreshed by calling the `lgrp_fini` function with the old cookie, followed by calling `lgrp_init` to generate a new cookie.

The `lgrp_cookie_stale` function returns EINVAL when the given cookie is invalid.

Using `lgrp_view`

The lgrp_view(3LGRP) function determines the view with which a given lgroup hierarchy snapshot was taken.

```
#include <sys/lgrp_user.h>
lgrp_view_t lgrp_view(lgrp_cookie_t cookie);
```

The `lgrp_view` function takes a cookie that represents a snapshot of the lgroup hierarchy and returns the snapshot's view of the lgroup hierarchy. Snapshots that are taken with the view `LGRP_VIEW_CALLER` contain only the resources that are available to the calling thread. Snapshots that are taken with the view `LGRP_VIEW_OS` contain all the resources that are available to the operating system.

The `lgrp_view` function returns EINVAL when the given cookie is invalid.

Using `lgrp_nlgrps`

The lgrp_nlgrps(3LGRP) function returns the number of locality groups in the system. If a system has only one locality group, memory placement optimizations have no effect.

```
#include <sys/lgrp_user.h>
int lgrp_nlgrps(lgrp_cookie_t cookie);
```

The `lgrp_nlgrps` function takes a cookie that represents a snapshot of the lgroup hierarchy and returns the number of lgroups available in the hierarchy.

The `lgrp_nlgrps` function returns EINVAL when the cookie is invalid.

Using `lgrp_root`

The lgrp_root(3LGRP) function returns the root lgroup ID.

```
#include <sys/lgrp_user.h>
lgrp_id_t lgrp_root(lgrp_cookie_t cookie);
```

The `lgrp_root` function takes a cookie that represents a snapshot of the lgroup hierarchy and returns the root lgroup ID.

Using `lgrp_parents`

The lgrp_parents(3LGRP) function takes a cookie that represents a snapshot of the lgroup hierarchy and returns the number of parent lgroups for the specified lgroup.

```
#include <sys/lgrp_user.h>
int lgrp_parents(lgrp_cookie_t cookie, lgrp_id_t child,
                 lgrp_id_t *lgrp_array, uint_t lgrp_array_size);
```

If `lgrp_array` is not `NULL` and the value of `lgrp_array_size` is not zero, the `lgrp_parents` function fills the array with parent lgroup IDs until the array is full or all parent lgroup IDs are in the array. The root lgroup has zero parents. When the `lgrp_parents` function is called for the root lgroup, `lgrp_array` is not filled in.

The `lgrp_parents` function returns EINVAL when the cookie is invalid. The `lgrp_parents` function returns ESRCH when the specified lgroup ID is not found.

Using `lgrp_children`

The lgrp_children(3LGRP) function takes a cookie that represents the calling thread's snapshot of the lgroup hierarchy and returns the number of child lgroups for the specified lgroup.

```
#include <sys/lgrp_user.h>
int lgrp_children(lgrp_cookie_t cookie, lgrp_id_t parent,
                  lgrp_id_t *lgrp_array, uint_t lgrp_array_size);
```

If `lgrp_array` is not `NULL` and the value of `lgrp_array_size` is not zero, the `lgrp_children` function fills the array with child lgroup IDs until the array is full or all child lgroup IDs are in the array.

The `lgrp_children` function returns EINVAL when the cookie is invalid. The `lgrp_children` function returns ESRCH when the specified lgroup ID is not found.

1.5 Locality Group Contents

The following APIs retrieve information about the contents of a given lgroup.

The lgroup hierarchy organizes the domain's resources to simplify the process of locating the nearest resource. Leaf lgroups are defined with resources that have the least latency. Each of the successive ancestor lgroups of a given leaf lgroup contains the next nearest resources to its child lgroup. The root lgroup contains all of the resources that are in the domain.

The resources of a given lgroup are contained directly within that lgroup or indirectly within the leaf lgroups that the given lgroup encapsulates. Leaf lgroups directly contain their resources and do not encapsulate any other lgroups.

Using `lgrp_resources`

The `lgrp_resources` function returns the number of resources contained in a specified lgroup.

```
#include <sys/lgrp_user.h>
int lgrp_resources(lgrp_cookie_t cookie, lgrp_id_t lgrp, lgrp_id_t *lgrpids,
                   uint_t count, lgrp_rsrc_t type);
```

The lgrp_resources function takes a cookie that represents a snapshot of the lgroup hierarchy. That cookie is obtained from the lgrp_init function. The lgrp_resources function returns the number of resources that are in the lgroup with the ID that is specified by the value of the lgrp argument. The lgrp_resources function represents the resources with a set of lgroups that directly contain CPU or memory resources. The lgrp_rsrc_t argument can have the following two values:

LGRP_RSRC_CPU
The lgrp_resources function returns the number of CPU resources.

LGRP_RSRC_MEM
The lgrp_resources function returns the number of memory resources.

When the value passed in the lgrpids[] argument is not null and the count argument is not zero, the lgrp_resources function stores lgroup IDs in the lgrpids[] array. The number of lgroup IDs stored in the array can be up to the value of the count argument.

The lgrp_resources function returns EINVAL when the specified cookie, lgroup ID, or type are not valid. The lgrp_resources function returns ESRCH when the function does not find the specified lgroup ID.

Using lgrp_cpus

The lgrp_cpus(3LGRP) function takes a cookie that represents a snapshot of the lgroup hierarchy and returns the number of CPUs in a given lgroup.

```
#include <sys/lgrp_user.h>
int lgrp_cpus(lgrp_cookie_t cookie, lgrp_id_t lgrp, processorid_t *cpuids,
              uint_t count, int content);
```

If the *cpuid[]* argument is not NULL and the CPU count is not zero, the lgrp_cpus function fills the array with CPU IDs until the array is full or all the CPU IDs are in the array.

The *content* argument can have the following two values:

LGRP_CONTENT_ALL
The lgrp_cpus function returns IDs for the CPUs in this lgroup and this lgroup's descendants.

LGRP_CONTENT_DIRECT
The lgrp_cpus function returns IDs for the CPUs in this lgroup only.

The lgrp_cpus function returns EINVAL when the cookie, lgroup ID, or one of the flags is not valid. The lgrp_cpus function returns ESRCH when the specified lgroup ID is not found.

Using lgrp_mem_size

The lgrp_mem_size(3LGRP) function takes a cookie that represents a snapshot of the lgroup hierarchy and returns the size of installed or free memory in the given lgroup. The lgrp_mem_size function reports memory sizes in bytes.

```
#include <sys/lgrp_user.h>
lgrp_mem_size_t lgrp_mem_size(lgrp_cookie_t cookie, lgrp_id_t lgrp,
                              int type, int content)
```

The *type* argument can have the following two values:

LGRP_MEM_SZ_FREE
The lgrp_mem_size function returns the amount of free memory in bytes.

LGRP_MEM_SZ_INSTALLED
The lgrp_mem_size function returns the amount of installed memory in bytes.

The *content* argument can have the following two values:

LGRP_CONTENT_ALL
The lgrp_mem_size function returns the amount of memory in this lgroup and this lgroup's descendants.

LGRP_CONTENT_DIRECT
The lgrp_mem_size function returns the amount of memory in this lgroup only.

The lgrp_mem_size function returns EINVAL when the cookie, lgroup ID, or one of the flags is not valid. The lgrp_mem_size function returns ESRCH when the specified lgroup ID is not found.

1.6 Locality Group Characteristics

The following API retrieves information about the characteristics of a given lgroup.

Using lgrp_latency_cookie

The lgrp_latency(3LGRP) function returns the latency between a CPU in one lgroup to the memory in another lgroup.

```
#include <sys/lgrp_user.h>
int lgrp_latency_cookie(lgrp_cookie_t cookie, lgrp_id_t from, lgrp_id_t to.
                        lat_between_t between);
```

The lgrp_latency_cookie function takes a cookie that represents a snapshot of the lgroup hierarchy. The lgrp_init function creates this cookie. The lgrp_latency_cookie function returns a value that represents the latency between a hardware resource in the lgroup given by the value of the *from* argument and a hardware resource in the lgroup given by the value of the *to* argument. If both arguments point to the same lgroup, the lgrp_latency_cookie function returns the latency value within that lgroup.

Note
The latency value returned by the lgrp_latency_cookie function is defined by the operating system and is platform-specific. This value does not necessarily represent the actual latency between hardware devices. Use this value only for comparison within one domain.

When the value of the *between* argument is LGRP_LAT_CPU_TO_MEM, the lgrp_latency_cookie function measures the latency from a CPU resource to a memory resource.

The lgrp_latency_cookie function returns EINVAL when the lgroup ID is not valid. When the lgrp_latency_cookie function does not find the specified lgroup ID, the "from" lgroup does not contain any CPUs, or the "to" lgroup does not have any memory, the lgrp_latency_cookie function returns ESRCH.

1.7 Locality Groups and Thread and Memory Placement

This section discusses the APIs used to discover and affect thread and memory placement with respect to lgroups.

- The lgrp_home(3LGRP) function is used to discover thread placement.
- The meminfo(2) system call is used to discover memory placement.
- The MADV_ACCESS flags to the madvise(3C) function are used to affect memory allocation among lgroups.
- The lgrp_affinity_set(3LGRP) function can affect thread and memory placement by setting a thread's affinity for a given lgroup.
- The affinities of an lgroup may specify an order of preference for lgroups from which to allocate resources.
- The kernel needs information about the likely pattern of an application's memory use in order to allocate memory resources efficiently.
- The madvise function and its shared object analogue madv.so.1 provide this information to the kernel.
- A running process can gather memory usage information about itself by using the meminfo system call.

Using lgrp_home

The lgrp_home function returns the home lgroup for the specified process or thread.

```
#include <sys/lgrp_user.h>
lgrp_id_t lgrp_home(idtype_t idtype, id_t id);
```

The lgrp_home function returns EINVAL when the ID type is not valid. The lgrp_home function returns EPERM when the effective user of the calling process is not the superuser and the real or effective user ID of the calling process does not match the real or effective user ID of one of the threads. The lgrp_home function returns ESRCH when the specified process or thread is not found.

Using `madvise`

The `madvise` function advises the kernel that a region of user virtual memory in the range starting at the address specified in *`addr`* and with length equal to the value of the *`len`* parameter is expected to follow a particular pattern of use. The kernel uses this information to optimize the procedure for manipulating and maintaining the resources associated with the specified range. Use of the `madvise` function can increase system performance when used by programs that have specific knowledge of their access patterns over memory.

```
#include <sys/types.h>
#include <sys/mman.h>
int madvise(caddr_t addr, size_t len, int advice);
```

The `madvise` function provides the following flags to affect how a thread's memory is allocated among lgroups:

`MADV_ACCESS_DEFAULT`
This flag resets the kernel's expected access pattern for the specified range to the default.

`MADV_ACCESS_LWP`
This flag advises the kernel that the next LWP to touch the specified address range is the LWP that will access that range the most. The kernel allocates the memory and other resources for this range and the LWP accordingly.

`MADV_ACCESS_MANY`
This flag advises the kernel that many processes or LWPs will access the specified address range randomly across the system. The kernel allocates the memory and other resources for this range accordingly.

The `madvise` function can return the following values:

EAGAIN
Some or all of the mappings in the specified address range, from *`addr`* to *`addr`*+*`len`*, are locked for I/O.

EINVAL
The value of the *`addr`* parameter is not a multiple of the page size as returned by sysconf(3C), the length of the specified address range is less than or equal to zero, or the advice is invalid.

EIO An I/O error occurs while reading from or writing to the file system.

ENOMEM
Addresses in the specified address range are outside the valid range for the address space of a process or the addresses in the specified address range specify one or more pages that are not mapped.

ESTALE
The NFS file handle is stale.

Using `madv.so.1`

The `madv.so.1` shared object enables the selective configuration of virtual memory advice for launched processes and their descendants. To use the shared object, the following string must be present in the environment:

```
LD_PRELOAD=$LD_PRELOAD:madv.so.1
```

The `madv.so.1` shared object applies memory advice as specified by the value of the `MADV` environment variable. The `MADV` environment variable specifies the virtual memory advice to use for all heap, shared memory, and mmap regions in the process address space. This advice is applied to all created processes. The following values of the `MADV` environment variable affect resource allocation among lgroups:

`access_default`
This value resets the kernel's expected access pattern to the default.

`access_lwp`
This value advises the kernel that the next LWP to touch an address range is the LWP that will access that range the most. The kernel allocates the memory and other resources for this range and the LWP accordingly.

`access_many`
This value advises the kernel that many processes or LWPs will access memory randomly across the system. The kernel allocates the memory and other resources accordingly.

The value of the `MADVCFGFILE` environment variable is the name of a text file that contains one or more memory advice configuration entries in the form *`exec-name`*:*`advice-opts`*.

The value of *`exec-name`* is the name of an application or executable. The value of *`exec-name`* can be a full pathname, a base name, or a pattern string.

The value of *`advice-opts`* is of the form *`region`*=*`advice`*. The values of *`advice`* are the same as the values for the `MADV` environment variable. Replace *`region`* with any of the following legal values:

`madv`
Advice applies to all heap, shared memory, and mmap(2) regions in the process address space.

`heap`
The heap is defined to be the brk(2) area. Advice applies to the existing heap and to any additional heap memory allocated in the future.

`shm` Advice applies to shared memory segments. See shmat(2) for more information on shared memory operations.

`ism` Advice applies to shared memory segments that are using the `SHM_SHARE_MMU` flag. The `ism` option takes precedence over `shm`.

`dsm` Advice applies to shared memory segments that are using the `SHM_PAGEABLE` flag. The `dsm` option takes precedence over `shm`.

`mapshared`
Advice applies to mappings established by the `mmap` system call using the `MAP_SHARED` flag.

mapprivate
Advice applies to mappings established by the `mmap` system call using the `MAP_PRIVATE` flag.

mapanon
Advice applies to mappings established by the `mmap` system call using the `MAP_ANON` flag. The `mapanon` option takes precedence when multiple options apply.

The value of the `MADVERRFILE` environment variable is the name of the path where error messages are logged. In the absence of a `MADVERRFILE` location, the `madv.so.1` shared object logs errors by using syslog(3C) with a `LOG_ERR` as the severity level and `LOG_USER` as the facility descriptor.

Memory advice is inherited. A child process has the same advice as its parent. The advice is set back to the system default advice after a call to exec(2) unless a different level of advice is configured using the `madv.so.1` shared object. Advice is only applied to `mmap` regions explicitly created by the user program. Regions established by the run-time linker or by system libraries that make direct system calls are not affected.

`madv.so.1` Usage Examples

The following examples illustrate specific aspects of the `madv.so.1` shared object.

Example 1.2: Setting Advice for a Set of Applications

This configuration applies advice to all ISM segments for applications with exec names that begin with `foo`.

```
$ LD_PRELOAD=$LD_PRELOAD:madv.so.1
$ MADVCFGFILE=madvcfg
$ export LD_PRELOAD MADVCFGFILE
$ cat $MADVCFGFILE
        foo*:ism=access_lwp
```

Example 1.3: Excluding a Set of Applications From Advice

This configuration sets advice for all applications with the exception of **ls**.

```
$ LD_PRELOAD=$LD_PRELOAD:madv.so.1
$ MADV=access_many
$ MADVCFGFILE=madvcfg
$ export LD_PRELOAD MADV MADVCFGFILE
$ cat $MADVCFGFILE
        ls:
```

Example 1.4: Pattern Matching in a Configuration File

Because the configuration specified in `MADVCFGFILE` takes precedence over the value set in `MADV`, specifying * as the *exec-name* of the last configuration entry is equivalent to setting `MADV`. This example is equivalent to the previous example.

```
$ LD_PRELOAD=$LD_PRELOAD:madv.so.1
$ MADVCFGFILE=madvcfg
```

```
$ export LD_PRELOAD MADVCFGFILE
$ cat $MADVCFGFILE
        ls:
        *:madv=access_many
```

Example 1.5: Advice for Multiple Regions

This configuration applies one type of advice for `mmap` regions and different advice for heap and shared memory regions for applications whose `exec` names begin with `foo`.

```
$ LD_PRELOAD=$LD_PRELOAD:madv.so.1
$ MADVCFGFILE=madvcfg
$ export LD_PRELOAD MADVCFGFILE
$ cat $MADVCFGFILE
        foo*:madv=access_many,heap=sequential,shm=access_lwp
```

Using `meminfo`

The `meminfo` function gives the calling process information about the virtual memory and physical memory that the system has allocated to that process.

```
#include <sys/types.h>
#include <sys/mman.h>
int meminfo(const uint64_t inaddr[], int addr_count,
    const uint_t info_req[], int info_count, uint64_t outdata[],
    uint_t validity[]);
```

The `meminfo` function can return the following types of information:

`MEMINFO_VPHYSICAL`
The physical memory address corresponding to the given virtual address

`MEMINFO_VLGRP`
The lgroup to which the physical page corresponding to the given virtual address belongs

`MEMINFO_VPAGESIZE`
The size of the physical page corresponding to the given virtual address

`MEMINFO_VREPLCNT`
The number of replicated physical pages that correspond to the given virtual address

`MEMINFO_VREPL|n`
The *n*th physical replica of the given virtual address

`MEMINFO_VREPL_LGRP|n`
The lgroup to which the *n*th physical replica of the given virtual address belongs

`MEMINFO_PLGRP`
The lgroup to which the given physical address belongs

The `meminfo` function takes the following parameters:

inaddr
An array of input addresses.

addr_count
The number of addresses that are passed to `meminfo`.

info_req
An array that lists the types of information that are being requested.

info_count
The number of pieces of information that are requested for each address in the *`inaddr`* array.

outdata
An array where the `meminfo` function places the results. The array's size is equal to the product of the values of the *`info_req`* and *`addr_count`* parameters.

validity
An array of size equal to the value of the *`addr_count`* parameter. The *`validity`* array contains bitwise result codes. The *0*th bit of the result code evaluates the validity of the corresponding input address. Each successive bit in the result code evaluates the validity of the response to the members of the *`info_req`* array in turn.

The `meminfo` function returns EFAULT when the area of memory to which the *`outdata`* or *`validity`* arrays point cannot be written to. The `meminfo` function returns EFAULT when the area of memory to which the *`info_req`* or *`inaddr`* arrays point cannot be read from. The `meminfo` function returns EINVAL when the value of *`info_count`* exceeds 31 or is less than 1. The `meminfo` function returns EINVAL when the value of *`addr_count`* is less than zero.

Example 1.6: Use of `meminfo` to Print Out Physical Pages and Page Sizes Corresponding to a Set of Virtual Addresses

```
void
print_info(void **addrvec, int how_many)
{
        static const int info[] = {
                MEMINFO_VPHYSICAL,
                MEMINFO_VPAGESIZE};
        uint64_t * inaddr = alloca(sizeof(uint64_t) * how_many);
        uint64_t * outdata = alloca(sizeof(uint64_t) * how_many * 2;
        uint_t * validity = alloca(sizeof(uint_t) * how_many);

        int i;

        for (i = 0; i < how_many; i++)
                inaddr[i] = (uint64_t *)addr[i];

        if (meminfo(inaddr, how_many,  info,
                    sizeof (info)/ sizeof(info[0]),
                    outdata, validity) < 0)
                ...

        for (i = 0; i < how_many; i++) {
                if (validity[i] & 1 == 0)
                        printf("address 0x%llx not part of address
                                        space\n",
```

```
                        inaddr[i]);

                else if (validity[i] & 2 == 0)
                        printf("address 0x%llx has no physical page
                                associated with it\n",
                        inaddr[i]);

                else {
                        char buff[80];
                        if (validity[i] & 4 == 0)
                                strcpy(buff, "<Unknown>");
                        else
                                sprintf(buff, "%lld", outdata[i * 2 +
                                        1]);
                        printf("address 0x%llx is backed by physical
                                page 0x%llx of size %s\n",
                                inaddr[i], outdata[i * 2], buff);
                }
        }
}
```

Locality Group Affinity

The kernel assigns a thread to a locality group when the lightweight process (LWP) for that thread is created. That lgroup is called the thread's *home lgroup*. The kernel runs the thread on the CPUs in the thread's home lgroup and allocates memory from that lgroup whenever possible. If resources from the home lgroup are unavailable, the kernel allocates resources from other lgroups. When a thread has affinity for more than one lgroup, the operating system allocates resources from lgroups chosen in order of affinity strength. Lgroups can have one of three distinct affinity levels:

1. `LGRP_AFF_STRONG` indicates strong affinity. If this lgroup is the thread's home lgroup, the operating system avoids rehoming the thread to another lgroup if possible. Events such as dynamic reconfiguration, processor, offlining, processor binding, and processor set binding and manipulation might still result in thread rehoming.

2. `LGRP_AFF_WEAK` indicates weak affinity. If this lgroup is the thread's home lgroup, the operating system rehomes the thread if necessary for load balancing purposes.

3. `LGRP_AFF_NONE` indicates no affinity. If a thread has no affinity to any lgroup, the operating system assigns a home lgroup to the thread .

The operating system uses lgroup affinities as advice when allocating resources for a given thread. The advice is factored in with the other system constraints. Processor binding and processor sets do not change lgroup affinities, but might restrict the lgroups on which a thread can run.

Using `lgrp_affinity_get`

The lgrp_affinity_get(3LGRP) function returns the affinity that a LWP has for a given lgroup.

```
#include <sys/lgrp_user.h>
lgrp_affinity_t lgrp_affinity_get(idtype_t idtype, id_t id, lgrp_id_t lgrp);
```

The *idtype* and *id* arguments specify the LWP that the lgrp_affinity_get function examines. If the value of *idtype* is P_PID, the lgrp_affinity_get function gets the lgroup affinity for one of the LWPs in the process whose process ID matches the value of the *id* argument. If the value of *idtype* is P_LWPID, the lgrp_affinity_get function gets the lgroup affinity for the LWP of the current process whose LWP ID matches the value of the *id* argument. If the value of *idtype* is P_MYID, the lgrp_affinity_get function gets the lgroup affinity for the current LWP.

The lgrp_affinity_get function returns EINVAL when the given lgroup or ID type is not valid. The lgrp_affinity_get function returns EPERM when the effective user of the calling process is not the superuser and the ID of the calling process does not match the real or effective user ID of one of the LWPs. The lgrp_affinity_get function returns ESRCH when a given lgroup or LWP is not found.

Using lgrp_affinity_set

The lgrp_affinity_set(3LGRP) function sets the affinity that a LWP or set of LWPs have for a given lgroup.

```
#include <sys/lgrp_user.h>
int lgrp_affinity_set(idtype_t idtype, id_t id, lgrp_id_t lgrp,
                      lgrp_affinity_t affinity);
```

The *idtype* and *id* arguments specify the LWP or set of LWPs the lgrp_affinity_set function examines. If the value of *idtype* is P_PID, the lgrp_affinity_set function sets the lgroup affinity for all of the LWPs in the process whose process ID matches the value of the *id* argument to the affinity level specified in the *affinity* argument. If the value of *idtype* is P_LWPID, the lgrp_affinity_set function sets the lgroup affinity for the LWP of the current process whose LWP ID matches the value of the *id* argument to the affinity level specified in the *affinity* argument. If the value of *idtype* is P_MYID, the lgrp_affinity_set function sets the lgroup affinity for the current LWP or process to the affinity level specified in the *affinity* argument.

The lgrp_affinity_set function returns EINVAL when the given lgroup, affinity, or ID type is not valid. The lgrp_affinity_set function returns EPERM when the effective user of the calling process is not the superuser and the ID of the calling process does not match the real or effective user ID of one of the LWPs. The lgrp_affinity_set function returns ESRCH when a given lgroup or LWP is not found.

1.8 Examples of API Usage

This section contains code for example tasks that use the APIs that are described in this chapter.

Example 1.7: Move Memory to a Thread

The following code sample moves the memory in the address range between *addr* and *addr*+*len* near the next thread to touch that range.

```
#include <stdio.h>
#include <sys/mman.h>
#include <sys/types.h>
```

```
/*
 * Move memory to thread
 */
void
mem_to_thread(caddr_t addr, size_t len)
{
    if (madvise(addr, len, MADV_ACCESS_LWP) < 0)
        perror("madvise");
}
```

Example 1.8: Move a Thread to Memory

This sample code uses the `meminfo` function to determine the lgroup of the physical memory backing the virtual page at the given address. The sample code then sets a strong affinity for that lgroup in an attempt to move the current thread near that memory.

```
#include <stdio.h>
#include <sys/lgrp_user.h>
#include <sys/mman.h>
#include <sys/types.h>


/*
 * Move a thread to memory
 */
int
thread_to_memory(caddr_t va)
{
    uint64_t    addr;
    ulong_t     count;
    lgrp_id_t   home;
    uint64_t    lgrp;
    uint_t      request;
    uint_t      valid;

    addr = (uint64_t)va;
    count = 1;
    request = MEMINFO_VLGRP;
    if (meminfo(&addr, 1, &request, 1, &lgrp, &valid) != 0) {
        perror("meminfo");
        return (1);
    }

    if (lgrp_affinity_set(P_LWPID, P_MYID, lgrp, LGRP_AFF_STRONG) != 0) {
        perror("lgrp_affinity_set");
        return (2);
    }

    home = lgrp_home(P_LWPID, P_MYID);
    if (home == -1) {
        perror ("lgrp_home");
        return (3);
    }

    if (home != lgrp)
        return (-1);

    return (0);
```

```
}
```

Example 1.9: Walk the lgroup Hierarchy

The following sample code walks through and prints out the lgroup hierarchy.

```
#include <stdio.h>
#include <stdlib.h>
#include <sys/lgrp_user.h>
#include <sys/types.h>


/*
 * Walk and print lgroup hierarchy from given lgroup
 * through all its descendants
 */
int
lgrp_walk(lgrp_cookie_t cookie, lgrp_id_t lgrp, lgrp_content_t content)
{
    lgrp_affinity_t    aff;
    lgrp_id_t          *children;
    processorid_t      *cpuids;
    int                i;
    int                ncpus;
    int                nchildren;
    int                nparents;
    lgrp_id_t          *parents;
    lgrp_mem_size_t    size;

    /*
     * Print given lgroup, caller's affinity for lgroup,
     * and desired content specified
     */
    printf("LGROUP #%d:\n", lgrp);

    aff = lgrp_affinity_get(P_LWPID, P_MYID, lgrp);
    if (aff == -1)
        perror ("lgrp_affinity_get");
    printf("\tAFFINITY: %d\n", aff);

    printf("CONTENT %d:\n", content);

    /*
     * Get CPUs
     */
    ncpus = lgrp_cpus(cookie, lgrp, NULL, 0, content);
    printf("\t%d CPUS: ", ncpus);
    if (ncpus == -1) {
        perror("lgrp_cpus");
        return (-1);
    } else if (ncpus > 0) {
        cpuids = malloc(ncpus * sizeof (processorid_t));
        ncpus = lgrp_cpus(cookie, lgrp, cpuids, ncpus, content);
                if (ncpus == -1) {
            free(cpuids);
                        perror("lgrp_cpus");
            return (-1);
        }
```

```
    for (i = 0; i < ncpus; i++)
        printf("%d ", cpuids[i]);
    free(cpuids);
}
printf("\n");

/*
 * Get memory size
 */
printf("\tMEMORY: ");
size = lgrp_mem_size(cookie, lgrp, LGRP_MEM_SZ_INSTALLED, content);
if (size == -1) {
    perror("lgrp_mem_size");
    return (-1);
}
printf("installed bytes 0x%llx, ", size);
size = lgrp_mem_size(cookie, lgrp, LGRP_MEM_SZ_FREE, content);
    if (size == -1) {
    perror("lgrp_mem_size");
    return (-1);
}
printf("free bytes 0x%llx\n", size);

/*
 * Get parents
 */
nparents = lgrp_parents(cookie, lgrp, NULL, 0);
printf("\t%d PARENTS: ", nparents);
if (nparents == -1) {
    perror("lgrp_parents");
    return (-1);
} else if (nparents > 0) {
    parents = malloc(nparents * sizeof (lgrp_id_t));
    nparents = lgrp_parents(cookie, lgrp, parents, nparents);
            if (nparents == -1) {
        free(parents);
                    perror("lgrp_parents");
        return (-1);
            }
    for (i = 0; i < nparents; i++)
        printf("%d ", parents[i]);
    free(parents);
}
printf("\n");

/*
 * Get children
 */
nchildren = lgrp_children(cookie, lgrp, NULL, 0);
printf("\t%d CHILDREN: ", nchildren);
if (nchildren == -1) {
    perror("lgrp_children");
    return (-1);
} else if (nchildren > 0) {
    children = malloc(nchildren * sizeof (lgrp_id_t));
    nchildren = lgrp_children(cookie, lgrp, children, nchildren);
            if (nchildren == -1) {
        free(children);
                    perror("lgrp_children");
        return (-1);
```

```
                }
        printf("Children: ");
        for (i = 0; i < nchildren; i++)
            printf("%d ", children[i]);
        printf("\n");

        for (i = 0; i < nchildren; i++)
            lgrp_walk(cookie, children[i], content);

        free(children);
    }
    printf("\n");

    return (0);
}
```

Example 1.10: Find the Closest lgroup With Available Memory Outside a Given lgroup

```
#include <stdio.h>
#include <stdlib.h>
#include <sys/lgrp_user.h>
#include <sys/types.h>


#define INT_MAX 2147483647


/*
 * Find next closest lgroup outside given one with available memory
 */
lgrp_id_t
lgrp_next_nearest(lgrp_cookie_t cookie, lgrp_id_t from)
{
    lgrp_id_t          closest;
    int                i;
    int                latency;
    int                lowest;
    int                nparents;
    lgrp_id_t          *parents;
    lgrp_mem_size_t    size;


    /*
     * Get number of parents
     */
    nparents = lgrp_parents(cookie, from, NULL, 0);
    if (nparents == -1) {
        perror("lgrp_parents");
        return (LGRP_NONE);
    }

    /*
     * No parents, so current lgroup is next nearest
     */
    if (nparents == 0) {
        return (from);
    }
```

```
    /*
     * Get parents
     */
    parents = malloc(nparents * sizeof (lgrp_id_t));
    nparents = lgrp_parents(cookie, from, parents, nparents);
    if (nparents == -1) {
        perror("lgrp_parents");
        free(parents);
        return (LGRP_NONE);
        }

    /*
     * Find closest parent (ie. the one with lowest latency)
     */
    closest = LGRP_NONE;
    lowest = INT_MAX;
    for (i = 0; i < nparents; i++) {
        lgrp_id_t   lgrp;

        /*
         * See whether parent has any free memory
         */
        size = lgrp_mem_size(cookie, parents[i], LGRP_MEM_SZ_FREE,
            LGRP_CONTENT_ALL);
        if (size > 0)
            lgrp = parents[i];
        else {
            if (size == -1)
                perror("lgrp_mem_size");

            /*
             * Find nearest ancestor if parent doesn't
             * have any memory
             */
            lgrp = lgrp_next_nearest(cookie, parents[i]);
            if (lgrp == LGRP_NONE)
                continue;
        }

        /*
         * Get latency within parent lgroup
         */
        latency = lgrp_latency_cookie(lgrp, lgrp);
        if (latency == -1) {
            perror("lgrp_latency_cookie");
            continue;
        }

        /*
         * Remember lgroup with lowest latency
         */
        if (latency < lowest) {
            closest = lgrp;
            lowest = latency;
        }
    }

    free(parents);
    return (closest);
}
```

```
/*
 * Find lgroup with memory nearest home lgroup of current thread
 */
lgrp_id_t
lgrp_nearest(lgrp_cookie_t cookie)
{
    lgrp_id_t   home;
    longlong_t  size;

    /*
     * Get home lgroup
     */
    home = lgrp_home(P_LWPID, P_MYID);

    /*
     * See whether home lgroup has any memory available in its hierarchy
     */
    size = lgrp_mem_size(cookie, home, LGRP_MEM_SZ_FREE,
        LGRP_CONTENT_ALL);
    if (size == -1)
        perror("lgrp_mem_size");

    /*
     * It does, so return the home lgroup.
     */
    if (size > 0)
        return (home);

    /*
     * Otherwise, find next nearest lgroup outside of the home.
     */
    return (lgrp_next_nearest(cookie, home));
}
```

Example 1.11: Find Nearest lgroup With Free Memory

This example code finds the nearest lgroup with free memory to a given thread's home lgroup.

```
lgrp_id_t
lgrp_nearest(lgrp_cookie_t cookie)
{
        lgrp_id_t         home;
        longlong_t        size;

        /*
         * Get home lgroup
         */

        home = lgrp_home();

        /*
         * See whether home lgroup has any memory available in its hierarchy
         */

        if (lgrp_mem_size(cookie, lgrp, LGRP_MEM_SZ_FREE,
            LGRP_CONTENT_ALL, &size) == -1)
```

```
                perror("lgrp_mem_size");

        /*
         * It does, so return the home lgroup.
         */

        if (size > 0)
                return (home);

        /*
         * Otherwise, find next nearest lgroup outside of the home.
         */

        return (lgrp_next_nearest(cookie, home));
}
```

Chapter 2

MPO Observability Tools

This chapter describes the tools that are available to use the MPO functionality that is available in the illumos operating system.

This chapter discusses the following topics:

- Section 2.1 describes the tool that applies rules that define the memory use of a process.
- Section 2.2 describes the tool that can display and set a thread's affinity for a locality group.
- Section 2.3 prints information about the lgroup hierarchy, contents, and characteristics.
- Section 2.4 describes a Perl interface to the locality group API that is described in Chapter 1.

2.1 The `pmadvise` utility

The `pmadvise` utility applies rules to a process that define how that process uses memory. The `pmadvise` utility applies the rules, called *advice*, to the process with the madvise(3C) tool. This tool can apply advice to a specific subrange of locations in memory at a specific time. By contrast, the madv.so.1(1) tool applies the advice throughout the execution of the target program to all segments of a specified type.

The `pmadvise` utility has the following options:

`-f` This option takes control of the target process. This option overrides the control of any other process. See the proc(1) manual page.

`-o` This option specifies the advice to apply to the target process. Specify the advice in this format:

```
private=advice
shared=advice
heap=advice
stack=advice
address:length=advice
```

The value of the *`advice`* term can be one of the following:

```
normal
random
sequential
willneed
dontneed
free
access_lwp
access_many
access_default
```

You can specify an address and length to specify the subrange where the advice applies. Specify the address in hexadecimal notation and the length in bytes.

If you do not specify the length and the starting address refers to the start of a segment, the `pmadvise` utility applies the advice to that segment. You can qualify the length by adding the letters K, M, G, T, P, or E to specify kilobytes, megabytes, gigabytes, terabytes, or exabytes, respectively.

`-v` This option prints verbose output in the style of the pmap(1) tool that shows the value and locations of the advice rules currently in force.

The `pmadvise` tool attempts to process all legal options. When the pmadvise tool attempts to process an option that specifies an illegal address range, the tool prints an error message and skips that option. When the `pmadvise` tool finds a syntax error, it quits without processing any options and prints a usage message.

When the advice for a specific region conflicts with the advice for a more general region, the advice for the more specific region takes precedence. Advice that specifies a particular address range has precedence over advice for the heap and stack regions, and advice for the heap and stack regions has precedence over advice for private and shared memory.

The advice rules in each of the following groups are mutually exclusive from other advice rules within the same group:

```
MADV_NORMAL, MADV_RANDOM, MADV_SEQUENTIAL
MADV_WILLNEED, MADV_DONTNEED, MADV_FREE
MADV_ACCESS_DEFAULT, MADV_ACCESS_LWP, MADV_ACCESS_MANY
```

2.2 The `plgrp` tool

The `plgrp` utility can display or set the home lgroup and lgroup affinities for one or more processes, threads, or lightweight processes (LWPs). The system assigns a home lgroup to each thread on creation. When the system allocates a CPU or memory resource to a thread, it searches the lgroup hierarchy from the thread's home lgroup for the nearest available resources to the thread's home.

The system chooses a home lgroup for each thread. The thread's affinity for its home lgroup is initially set to none, or no affinity. When a thread sets an affinity for an lgroup in its processor set that is higher than the thread's affinity for its home lgroup, the system moves the thread to that lgroup. The system does not move threads that are bound to a CPU. The system rehomes a thread to the lgroup in its processor set that has the highest affinity when the thread's affinity for its home lgroup is removed (set to none).

For a full description of the different levels of lgroup affinity and their semantics, see the lgrp_affinity_set(3LGRP) manual page.

The `plgrp` tool supports the following options:

-a ***lgroup list***
This option displays the affinities of the processes or threads that you specify for the lgroups in the list.

-A ***lgroup list*****/none|weak|strong[,...]**
This option sets the affinity of the processes or threads that you specify for the lgroups in the list. You can use a comma separated list of *lgroup*/*affinity* assignments to set several affinities at once.

-F This option takes control of the target process. This option overrides the control of any other process. See the proc(1) manual page.

-h This option returns the home lgroup of the processes or threads that you specify. This is the default behavior of the plgrp tool when you do not specify any options.

-H ***lgroup list***
This option sets the home lgroup of the processes or threads that you specify. This option sets a strong affinity for the listed lgroup. If you specify more than one lgroup, the plgrp utility will attempt to home the threads to the lgroups in a round robin fashion.

Specifying Lgroups

The value of the *lgroup list* variable is a comma separated list of one or more of the following attributes:

- lgroup ID
- Range of lgroup IDs, specified as *start lgroup ID-end lgroup ID*
- all
- root
- leaves

The all keyword represents all of the lgroup IDs in the system. The root keyword represents the ID of the root lgroup. The leaves keyword represents the IDs of all of the leaf lgroups. A leaf lgroup is an lgroup that does not have any children.

Specifying Process and Thread Arguments

The plgrp utility takes one or more space-separated processes or threads as arguments. You can specify processes and threads in a the same syntax that the proc(1) tools use. You can specify a process ID as an integer, with the syntax *pid* or /proc/*pid*. You can use shell expansions with the /proc/*pid* syntax. When you give a process ID alone, the arguments to the plgrp utility include all of the threads of that process.

You can specify a thread explicitly by specifying the process ID and thread ID with the syntax *pid*/*lwpid*. You can specify multiple threads of a process by defining ranges with can be selected at once by using the - character to define a range, or with a comma-separated list. To specify threads 1, 2, 7, 8, and 9 of a process whose process ID is *pid*, use the syntax *pid*/1,2,7-9.

2.3 The `lgrpinfo` Tool

The `lgrpinfo` tool prints information about the lgroup hierarchy, contents, and characteristics. The `lgrpinfo` tool is a Perl script that requires the `Solaris::Lgrp` module. This tool uses the liblgrp(3LIB) API to get the information from the system and displays it in the human-readable form.

The `lgrpinfo` tool prints general information about all of the lgroups in the system when you call it without any arguments. When you pass lgroup IDs to the `lgrpinfo` tool at the command line, the tool returns information about the lgroups that you specify. You can specify lgroups with their lgroup IDs or with one of the following keywords:

`all` This keyword specifies all lgroups and is the default behavior.

`root`
This keyword specifies the root lgroup.

`leaves`
This keyword specifies all of the leaf lgroups. A leaf lgroup is an lgroup that has no children in the lgroup hierarchy.

`intermediate`
This keyword specifies all of the intermediate lgroups. An intermediate lgroup is an lgroup that has a parent and children.

When the `lgrpinfo` tool receives an invalid lgroup ID, the tool prints a message with the invalid ID and continues processing any other lgroups that are passed in the command line. When the `lgrpinfo` tool finds no valid lgroups in the arguments, it exits with a status of `2`.

Options for the `lgrpinfo` Tool

When you call the `lgrpinfo` tool without any arguments, the tool's behavior is equivalent to using the options `-celmrt all`. The valid options for the `lgrpinfo` tool are:

`-a` This option prints the topology, CPU, memory, load and latency information for the specified lgroup IDs. This option combines the behaviors of the `-tcemrlL` options, unless you also specify the `-T` option. When you specify the `-T` option, the behavior of the `-a` option does not include the behavior of the `-t` option.

`-c` This option prints the CPU information.

`-C` This option replaces each lgroup in the list with its children. You cannot combine this option with the `-P` or `-T` options. When you do not specify any arguments, the tool applies this option to all lgroups.

`-e` This option prints lgroup load averages for leaf lgroups.

`-G` This option prints the OS view of the lgroup hierarchy. The tool's default behavior displays the caller's view of the lgroup hierarchy. The caller's view only includes the resources that the caller can use. See the lgrp_init(3LGRP) manual page for more details on the OS and caller's view.

`-h` This option prints the help message for the tool.

-I This option prints only IDs that match the IDs you specify. You can combine this option with the -c, -G, -C, or -P options. When you specify the -c option, the tool prints the list of CPUs that are in all of the matching lgroups. When you do not specify the -c option, the tool displays the IDs for the matching lgroups. When you do not specify any arguments, the tool applies this option to all lgroups.

-l This option prints information about lgroup latencies. The latency value given for each lgroup is defined by the operating system and is platform-specific. It can only be used for relative comparison of lgroups on the running system. It does not necessarily represent the actual latency between hardware devices and may not be applicable across platforms.

-L This option prints the lgroup latency table. This table shows the relative latency from each lgroup to each of the other lgroups.

-m This option prints memory information. The tool reports memory sizes in the units that give a size result in the integer range from 0 to 1023. You can override this behavior by using the -u option. The tool will only display fractional results for values smaller than 10.

-P This option replaces each lgroup in the list with its parent or parents. You cannot combine this option with the -C or -T options. When you do not specify any arguments, the tool applies this option to all lgroups.

-r This option prints information about lgroup resources. When you specify the -T option, the tool displays information about the resources of the intermediate lgroups only.

-t This option prints information about lgroup topology.

-T This option prints the lgroup topology of a system graphically, as a tree. You can only use this option with the -a, -c, -e, -G, -l, -L, -m, -r, and -u options. To restrict the output to intermediate lgroups, use the -r option. Omit the -t option when you combine the -T option with the -a option. This option does not print information for the root lgroup unless it is the only lgroup.

-u*units*
This option specifies memory units. The value of the *units* argument can be b, k, m, g, t, p, or e for bytes, kilobytes, megabytes, gigabytes, terabytes, petabytes, or exabytes, respectively.

2.4 The Solaris::lgrp Module

This Perl module provides a Perl interface to the lgroup APIs that are in liblgrp. This interface provides a way to traverse the lgroup hierarchy, discover its contents and characteristics, and set a thread's affinity for an lgroup. The module gives access to various constants and functions defined in the lgrp_user.h header file. The module provides the procedural interface and the object interface to the library.

The default behavior of this module does not export anything. You can use the following tags to selectively import the constants and functions that are defined in this module:

:LGRP_CONSTANTS
LGRP_AFF_NONE, LGRP_AFF_STRONG, LGRP_AFF_WEAK, LGRP_CONTENT_DIRECT, LGRP_CONTENT_HIERARCHY, LGRP_MEM_SZ_FREE, LGRP_MEM_SZ_INSTALLED, LGRP_VER_CURRENT, LGRP_VER_NONE, LGRP_VIEW_CALLER, LGRP_VIEW_OS, LGRP_NONE, LGRP_RSRC_CPU, LGRP_RSRC_MEM, LGRP_CONTENT_ALL, LGRP_LAT_CPU_TO_MEM

:PROC_CONSTANTS
P_PID, P_LWPID, P_MYID

:CONSTANTS
:LGRP_CONSTANTS, :PROC_CONSTANTS

:FUNCTIONS
lgrp_affinity_get, lgrp_affinity_set, lgrp_children, lgrp_cookie_stale, lgrp_cpus, lgrp_fini, lgrp_home, lgrp_init, lgrp_latency, lgrp_latency_cookie, lgrp_mem_size, lgrp_nlgrps, lgrp_parents, lgrp_root, lgrp_version, lgrp_view, lgrp_resources, lgrp_lgrps, lgrp_leaves, lgrp_isleaf, lgrp_lgrps, lgrp_leaves.

:ALL
:CONSTANTS, :FUNCTIONS

The Perl module has the following methods:

- new
- cookie
- stale
- view
- root
- children
- parents
- nlgrps
- mem_size
- cpus
- isleaf
- resources
- version
- home
- affinity_get
- affinity_set
- lgrps
- leaves
- latency

You can export constants with the :CONSTANTS or :ALL tags. You can use any of the constants in the following list in Perl programs.

- LGRP_NONE
- LGRP_VER_CURRENT
- LGRP_VER_NONE
- LGRP_VIEW_CALLER
- LGRP_VIEW_OS
- LGRP_AFF_NONE
- LGRP_AFF_STRONG
- LGRP_AFF_WEAK
- LGRP_CONTENT_DIRECT
- LGRP_CONTENT_HIERARCHY
- LGRP_MEM_SZ_FREE
- LGRP_MEM_SZ_INSTALLED
- LGRP_RSRC_CPU
- LGRP_RSRC_MEM
- LGRP_CONTENT_ALL
- LGRP_LAT_CPU_TO_MEM
- P_PID
- P_LWPID
- P_MYID

When an underlying library function fails, the functions in this module return either undef or an empty list. The module can use the following error codes:

EINVAL
The value supplied is not valid.

ENOMEM
There was not enough system memory to complete an operation.

ESRCH
The specified process or thread was not found.

EPERM
The effective user of the calling process does not have the appropriate privileges, and its real or effective user ID does not match the real or effective user ID of one of the threads.

Appendix A

Document License

A.1 Public Documentation License (PDL), Version 1.01

1.0 DEFINITIONS.

1.1. "Commercial Use" means distribution or otherwise making the Documentation available to a third party.

1.2. "Contributor" means a person or entity who creates or contributes to the creation of Modifications.

1.3. "Documentation" means the Original Documentation or Modifications or the combination of the Original Documentation and Modifications, in each case including portions thereof.

1.4. "Editable Form" means the preferred form of Documentation for making Modifications to such documentation. The Documentation can be in an electronic, compressed or archival form, provided the appropriate decompression or de-archiving software is widely available for no charge.

1.5. "Electronic Distribution Mechanism" means a mechanism generally accepted for the electronic transfer of data.

1.6. "Initial Writer" means the individual or entity identified as the Initial Writer in the notice required by the Appendix.

1.7. "Larger Work" means a work which combines Documentation or portions thereof with documentation or other writings not governed by the terms of this License.

1.8. "License" means this document.

1.9. "Modifications" means any addition to or deletion from the substance or structure of either the Original Documentation or any previous Modifications, such as a translation, abridgment, condensation, or any other form in which the Original Documentation or previous Modifications may be recast, transformed or adapted. A work consisting of editorial revisions, annotations, elaborations, and other modifications which, as a whole represent an original work of authorship, is a Modification. For example, when Documentation is released as a series of documents, a Modification is:

A. Any addition to or deletion from the contents of the Original Documentation or previous Modifications.

B. Any new documentation that contains any part of the Original Documentation or previous Modifications.

1.10. "Original Documentation" means documentation described as Original Documentation in the notice required by the Appendix, and which, at the time of its release under this License is not already Documentation governed by this License.

1.11. "You" (or "Your") means an individual or a legal entity exercising rights under, and complying with all of the terms of, this License or a future version of this License issued under Section 5.0 ("Versions of the License"). For legal entities, "You" includes any entity which controls, is controlled by, or is under common control with You. For purposes of this definition, "control" means (a) the power, direct or indirect, to cause the direction or management of such entity, whether by contract or otherwise, or (b) ownership of more than fifty percent (50%) of the outstanding shares or beneficial ownership of such entity.

2.0 LICENSE GRANTS.

2.1 INITIAL WRITER GRANT.

The Initial Writer hereby grants You a world-wide, royalty-free, non-exclusive license to use, reproduce, prepare Modifications of, compile, publicly perform, publicly display, demonstrate, market, disclose and distribute the Documentation in any form, on any media or via any Electronic Distribution Mechanism or other method now known or later discovered, and to sublicense the foregoing rights to third parties through multiple tiers of sublicensees in accordance with the terms of this License.

The license rights granted in this Section 2.1 ("Initial Writer Grant") are effective on the date Initial Writer first distributes Original Documentation under the terms of this License.

2.2. CONTRIBUTOR GRANT.

Each Contributor hereby grants You a world-wide, royalty-free, non-exclusive license to use, reproduce, prepare Modifications of, compile, publicly perform, publicly display, demonstrate, market, disclose and distribute the Documentation in any form, on any media or via any Electronic Distribution Mechanism or other method now known or later discovered, and to sublicense the foregoing rights to third parties through multiple tiers of sublicensees in accordance with the terms of this License.

The license rights granted in this Section 2.2 ("Contributor Grant") are effective on the date Contributor first makes Commercial Use of the Documentation.

3.0 DISTRIBUTION OBLIGATIONS.

3.1. APPLICATION OF LICENSE.

The Modifications which You create or to which You contribute are governed by the terms of this License, including without limitation Section 2.2 ("Contributor Grant"). The Documentation may be distributed only under the terms of this License or a future version of this License released in accordance with Section 5.0 ("Versions of the License"), and You must include a copy of this License with every copy of the Documentation You distribute. You may not offer or impose any terms that alter or restrict the applicable version of this License or the recipients' rights hereunder. However, You may include an additional document offering the additional rights described in Section 3.5 ("Required Notices").

3.2. AVAILABILITY OF DOCUMENTATION.

Any Modification which You create or to which You contribute must be made available publicly in Editable Form under the terms of this License via a fixed medium or an accepted Electronic Distribution Mechanism.

3.3. DESCRIPTION OF MODIFICATIONS.

All Documentation to which You contribute must identify the changes You made to create that Documentation and the date of any change. You must include a prominent statement that the Modification is derived, directly or indirectly, from Original Documentation provided by the Initial Writer and include the name of the Initial Writer in the Documentation or via an electronic link that describes the origin or ownership of the Documentation. The foregoing change documentation may be created by using an electronic program that automatically tracks changes to the Documentation. You must make all such changes and required information publicly available for at least five years following release of the changed Documentation.

3.4. INTELLECTUAL PROPERTY MATTERS.

Contributor represents that Contributor believes that Contributor's Modifications are Contributor's original creation(s), Contributor has sufficient rights to grant the rights conveyed by this License, or both of these statements are true.

3.5. REQUIRED NOTICES.

You must duplicate the notice in the Appendix in each file of the Documentation. If it is not possible to put such notice in a particular Documentation file due to its structure, then You must include such notice in a location (such as a relevant directory) where a reader would be likely to look for such a notice, for example, via a hyperlink in each file of the Documentation that takes the reader to a page that describes the origin and ownership of the Documentation. If You created one or more Modification(s) You may add your name as a Contributor to the notice described in the Appendix.

You must also duplicate this License in any Documentation file (or with a hyperlink in each file of the Documentation) where You describe recipients' rights or ownership rights.

You may choose to offer, and to charge a fee for, warranty, support, indemnity or liability obligations to one or more recipients of Documentation. However, You may do so only on Your own behalf, and not on behalf of the Initial Writer or any Contributor. You must make it absolutely clear than any such warranty, support, indemnity or liability obligation is offered by You alone, and You hereby agree to indemnify the Initial Writer and every Contributor for any liability incurred by the Initial Writer or such Contributor as a result of warranty, support, indemnity or liability terms You offer.

3.6. LARGER WORKS.

You may create a Larger Work by combining Documentation with other documents not governed by the terms of this License and distribute the Larger Work as a single product. In such a case, You must make sure the requirements of this License are fulfilled for the Documentation.

4.0 APPLICATION OF THIS LICENSE.

This License applies to Documentation to which the Initial Writer has attached this License and the notice in the Appendix.

5.0 TRADEMARKS.

This license does not grant permission to use the trade names, trademarks, service marks, logos, or product names of the Initial Writer or any Contributor, except as required for reasonable and customary use in describing the Origin of the Documentation and reproducing the content of any of the notices described in the Appendix.

6.0 VERSIONS OF THE LICENSE.

6.1. NEW VERSIONS.

Initial Writer may publish revised or new versions of the License from time to time. Each version will be given a distinguishing version number.

6.2. EFFECT OF NEW VERSIONS.

Once Documentation has been published under a particular version of the License, You may always continue to use it under the terms of that version. You may also choose to use such Documentation under the terms of any subsequent version of the License published by _____[Insert name of the foundation, company, Initial Writer, or whoever may modify this License]. No one other than _____ [Insert name of the foundation, company, Initial Writer, or whoever may modify this License] has the right to modify the terms of this License. Filling in the name of the Initial Writer, Original Documentation or Contributor in the notice described in the Appendix shall not be deemed to be Modifications of this License.

7.0 DISCLAIMER OF WARRANTY.

DOCUMENTATION IS PROVIDED UNDER THIS LICENSE ON AN "AS IS" BASIS, WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESSED OR IMPLIED, INCLUDING, WITHOUT LIMITATION, WARRANTIES THAT THE DOCUMENTATION IS FREE OF DEFECTS, MERCHANTABLE, FIT FOR A PARTICULAR PURPOSE OR NON-INFRINGING. THE ENTIRE RISK AS TO THE QUALITY, ACCURACY, AND PERFORMANCE OF THE DOCUMENTATION IS WITH YOU. SHOULD ANY DOCUMENTATION PROVE DEFECTIVE IN ANY RESPECT, YOU (NOT THE INITIAL WRITER OR ANY OTHER CONTRIBUTOR) ASSUME THE COST OF ANY NECESSARY SERVICING, REPAIR OR CORRECTION. THIS DISCLAIMER OF WARRANTY CONSTITUTES AN ESSENTIAL PART OF THIS LICENSE. NO USE OF ANY DOCUMENTATION IS AUTHORIZED HEREUNDER EXCEPT UNDER THIS DISCLAIMER.

8.0 TERMINATION.

This License and the rights granted hereunder will terminate automatically if You fail to comply with terms herein and fail to cure such breach within 30 days of becoming aware of the breach. All sublicenses to the Documentation which are properly granted shall survive any termination of this License. Provisions which, by their nature, must remain in effect beyond the termination of this License shall survive.

9.0 LIMITATION OF LIABILITY.

UNDER NO CIRCUMSTANCES AND UNDER NO LEGAL THEORY, WHETHER IN TORT (INCLUDING NEGLIGENCE), CONTRACT, OR OTHERWISE, SHALL THE INITIAL WRITER, ANY OTHER CONTRIBUTOR, OR ANY DISTRIBUTOR OF DOCUMENTATION, OR ANY SUPPLIER OF ANY OF SUCH PARTIES, BE LIABLE TO ANY PERSON FOR ANY DIRECT, INDIRECT, SPECIAL, INCIDENTAL, OR CONSEQUENTIAL DAMAGES OF ANY CHARACTER INCLUDING, WITHOUT LIMITATION, DAMAGES FOR LOSS OF GOODWILL, WORK STOPPAGE, COMPUTER FAILURE OR MALFUNCTION, OR ANY AND ALL OTHER DAMAGES OR LOSSES ARISING OUT OF OR RELATING TO THE USE OF THE DOCUMENTATION, EVEN IF SUCH PARTY SHALL HAVE BEEN INFORMED OF THE POSSIBILITY OF SUCH DAMAGES.

10.0 U.S. GOVERNMENT END USERS.

If Documentation is being acquired by or on behalf of the U.S. Government or by a U.S. Government prime contractor or subcontractor (at any tier), then the Government's rights in Documentation will be only as set forth in this Agreement; this is in accordance with 48 CFR 227.7201 through 227.7202-4 (for Department of Defense (DOD) acquisitions) and with 48 CFR 2.101 and 12.212 (for non-DOD acquisitions).

11.0 MISCELLANEOUS.

This License represents the complete agreement concerning the subject matter hereof. If any provision of this License is held to be unenforceable, such provision shall be reformed only to the extent necessary to make it enforceable. This License shall be governed by California law, excluding its conflict-of-law provisions. With respect to disputes or any litigation relating to this License, the losing party is responsible for costs, including without limitation, court costs and reasonable attorneys' fees and expenses. The application of the United Nations Convention on Contracts for the International Sale of Goods is expressly excluded. Any law or regulation which provides that the language of a contract shall be construed against the drafter shall not apply to this License.

A.2 Public Documentation License Notice

The contents of this Documentation are subject to the Public Documentation License Version 1.01 (the "License"); you may only use this Documentation if you comply with the terms of this License. A copy of the License is available at http://illumos.org/license/PDL

The Original Documentation is the Memory and Thread Placement Optimization Developer's Guide. The Initial Writer of the Original Documentation is Sun Microsystems Copyright (C) 2003-2009. All Rights Reserved. (Initial Writer contact(s): http://sun.com).

Contributor(s): Joyent, Inc. Portions created by Joyent, Inc. are Copyright (C) 2016. All Rights Reserved. (Contributor contact(s): http://joyent.com).

This documentation was derived from the source at illumos docbooks. For full changes as required by the PDL, please see the above URL.