Solaris: メモリーとスレッドの配置最適化開発者ガイド

Sun Microsystems, Inc.
4150 Network Circle
Santa Clara, CA 95054
U.S.A.

Part No: 820–1930–10
2007 年 6 月

原典: Solaris: Memory and Thread Placement Optimization Developer's Guide

Part No: 820-1691-10

Revision A

070612@17466

目次

はじめに

『メモリーとスレッドの配置最適化開発者ガイド』には、近傍性グループに関する説明と、Solaris オペレーティングシステムのコンピュータ資源を最適に使用するために利用できる技術に関する説明が記載されています。

対象読者

このマニュアルは、複数の CPU と均一でないメモリーアーキテクチャーを備えた環境でアプリケーションを作成する開発者を対象にしています。開発者は、このマニュアルで説明されているプログラミングインタフェースやプログラミングツールを使用して、システムの動作と資源割り当てを制御することができます。

第三者の関連する Web サイトの参照

このマニュアル内で参照している第三者の URL は、追加の関連情報を提供します。

注 – このマニュアル内で引用する第三者の Web サイトの可用性について Sun は責任を負いません。こうしたサイトやリソース上の、またはこれらを通じて利用可能な、コンテンツ、広告、製品、その他の素材について、Sun は推奨しているわけではなく、Sun はいかなる責任も負いません。こうしたサイトやリソース上の、またはこれらを経由して利用可能な、コンテンツ、製品、サービスを利用または信頼したことに伴って発生した (あるいは発生したと主張される) いかなる損害や損失についても、Sun は一切の責任を負いません。

マニュアル、サポート、およびトレーニング

Sun の Web サイトでは、次のサービスに関する情報も提供しています。

- マニュアル (http://jp.sun.com/documentation/)
- サポート (http://jp.sun.com/support/)
- トレーニング (http://jp.sun.com/training/)

表記上の規則

このマニュアルでは、次のような字体や記号を特別な意味を持つものとして使用します。

表 P–1 表記上の規則

字体または記号	意味	例
AaBbCc123	コマンド名、ファイル名、ディレクトリ名、画面上のコンピュータ出力、コード例を示します。	.login ファイルを編集します。 ls -a を使用してすべてのファイルを表示します。 system%
AaBbCc123	ユーザーが入力する文字を、画面上のコンピュータ出力と区別して示します。	system% **su** password:
AaBbCc123	変数を示します。実際に使用する特定の名前または値で置き換えます。	ファイルを削除するには、rm *filename* と入力します。
『 』	参照する書名を示します。	『コードマネージャ・ユーザーズガイド』を参照してください。
「 」	参照する章、節、ボタンやメニュー名、強調する単語を示します。	第 5 章「衝突の回避」を参照してください。 この操作ができるのは、「スーパーユーザー」だけです。
\	枠で囲まれたコード例で、テキストがページ行幅を超える場合に、継続を示します。	sun% **grep '^#define ** **XV_VERSION_STRING'**

コード例は次のように表示されます。

- C シェル

  machine_name% **command y|n** [*filename*]

- C シェルのスーパーユーザー

  machine_name# **command y|n** [*filename*]

- Bourne シェルおよび Korn シェル

  $ **command y|n** [*filename*]

- Bourne シェルおよび Korn シェルのスーパーユーザー

```
# command y|n [filename]
```

[ ] は省略可能な項目を示します。上記の例は、*filename* は省略してもよいことを示しています。

| は区切り文字 (セパレータ) です。この文字で分割されている引数のうち 1 つだけを指定します。

キーボードのキー名は英文で、頭文字を大文字で示します (例: Shift キーを押します)。ただし、キーボードによっては Enter キーが Return キーの動作をします。

ダッシュ (-) は 2 つのキーを同時に押すことを示します。たとえば、Ctrl-D は Control キーを押したまま D キーを押すことを意味します。

◆ ◆ ◆ 第 1 章

1 近傍性グループ API

この章では、近傍性グループとやりとりするために、アプリケーションで使用できる API について説明します。

次の内容について説明します。

- 10 ページの「近傍性グループの概要」では、近傍性グループによる抽象化について説明します。
- 12 ページの「インタフェースバージョンの確認」では、インタフェースに関する情報を提供する関数について説明します。
- 12 ページの「近傍性グループインタフェースの初期化」では、近傍性グループ階層を探索し、内容を検索するために使用するインタフェース部分の初期化およびシャットダウンを実行する関数呼び出しについて説明します。
- 14 ページの「近傍性グループ階層」では、近傍性グループ階層内を移動する関数呼び出しと、近傍性グループ階層の特性に関する情報を取得する関数について説明します。
- 16 ページの「近傍性グループの内容」では、近傍性グループの内容に関する情報を取り出す関数呼び出しについて説明します。
- 18 ページの「近傍性グループの特性」では、近傍性グループの特性に関する情報を取り出す関数呼び出しについて説明します。
- 19 ページの「近傍性グループ、スレッド、およびメモリー配置」では、スレッドのメモリー配置を制御する方法とその他のメモリー管理手法について説明します。
- 28 ページの「API の使用例」では、この章で説明した API を使用して、タスクを実行するコーディング例を示します。

近傍性グループの概要

メモリー共有型マルチプロセッサマシンには、複数の CPU が搭載されています。それぞれの CPU は、そのマシンのすべてのメモリーにアクセスできます。メモリー共有型マルチプロセッサには、CPU ごとに特定のメモリー領域に対して、より高速なアクセスを可能にするメモリーアーキテクチャーを採用しているものがあります。

そのようなメモリーアーキテクチャーのマシンで Solaris ソフトウェアを実行した場合、特定の CPU による特定のメモリー領域への最短アクセス時間に関するカーネル情報が提供されると、システムのパフォーマンスを向上させることができます。この情報を処理するために近傍性グループ (lgroup) による抽象化が導入されています。lgroup による抽象化は、メモリー配置の最適化 (MPO) 機能の一部です。

lgroup は CPU およびメモリーを模したデバイスの集合です。それぞれの集合内のデバイスは、決められた応答時間の間隔範囲で集合内の任意のデバイスにアクセスできます。応答時間間隔の値は、その lgroup 内のすべての CPU とすべてのメモリー間の最小の共通応答時間を表します。lgroup を定義する応答時間範囲は、その lgroup のメンバー間の最大応答時間を制限しません。応答時間範囲の値は、そのグループ内の CPU とメモリーのあらゆる組み合わせに共通する最小の応答時間です。

lgroup は階層構造になっています。lgroup 階層は、Directed Acyclic Graph (DAG) です。ツリー構造と似ていますが、lgroup は複数の親を持つことができます。ルート lgroup にはシステムのすべての資源が含まれており、子 lgroup を持つことができます。さらに、ルート lgroup にはシステムでもっとも高い応答時間値を持たせることができます。すべての子 lgroup の応答時間値は、ルートよりも低くなります。応答時間値はルートに近いほど高く、葉に近いほど低くなります。

すべての CPU がどのメモリー領域に対しても同じ時間でアクセスするコンピュータは、単一の lgroup として表すことができます (図 1–1 参照)。特定の CPU が特定のメモリー領域に対してほかの領域より高速にアクセスすることが可能となるコンピュータは、複数の lgroup を使用して表すことができます (図 1–2 参照)。

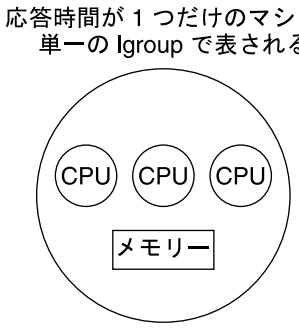

図 1–1　単一近傍性グループの模式図

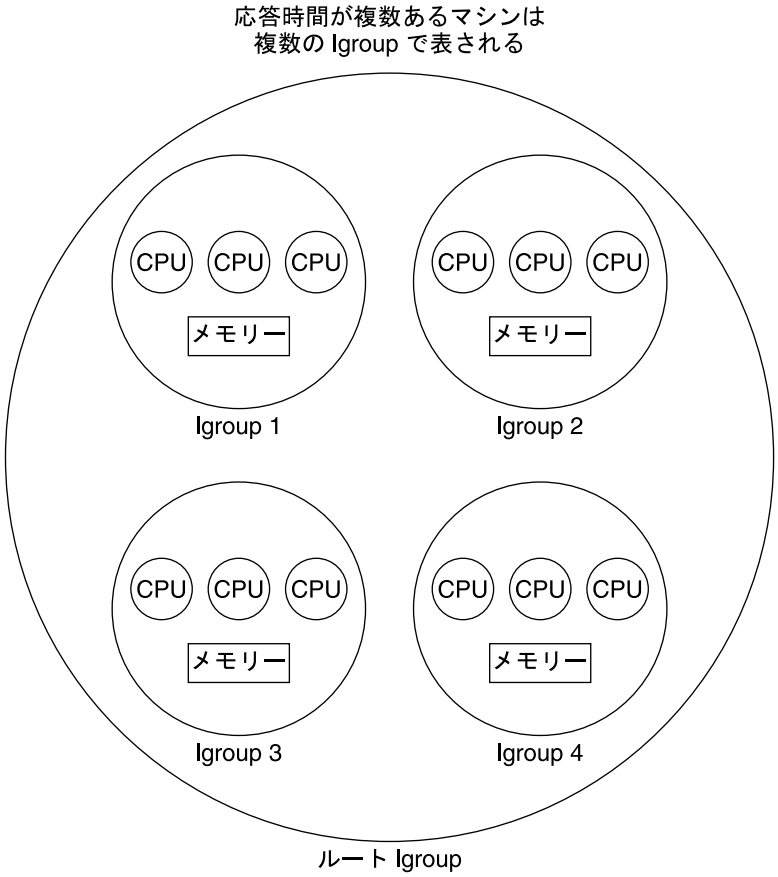

図 1–2　複数近傍性グループの模式図

組織化された lgroup 階層の導入によって、システムでもっとも近い資源を検索するタスクを簡略化します。各スレッドは、作成時にホーム lgroup に割り当てられます。オペレーティングシステムは、スレッドにホーム lgroup から資源を割り当てようとデフォルトで試みます。たとえば、Solaris カーネルが、あるスレッドのホーム lgroup にある CPU 上でそのスレッドを実行し、デフォルトでスレッドのホーム lgroup にそのスレッドのメモリーを割り当てるスケジュールを設定しようと試みます。必要な資源がスレッドのホーム lgroup で利用可能ではない場合には、ホーム lgroup の親から順番に lgroup 階層を探索し、次に近い位置にある資源を検索します。必要な資源がホーム lgroup の親で利用可能でない場合には、カーネルは引き続き lgroup 階層を探索し、そのホーム lgroup のさらに上位ノードの lgroup を順番に探索します。マシン内のすべての lgroup の最上位ノードに位置するのがルート lgroup で、これにはそのマシンのすべての資源が含まれます。

lgroup API は、監視と性能チューニングを目的とするアプリケーションのために lgroup の抽象化をエクスポートします。新しい API は、新規ライブラリ liblgrp に含まれています。アプリケーションは API を使用して、次の作業を実行できます。

- グループ階層の検索
- 指定された lgroup の内容と特性の検出
- lgroup でのスレッドとメモリー配置の操作

インタフェースバージョンの確認

lgroup API を使用する前に、lgrp_version(3LGRP) 関数を使用して、lgroup インタフェースがサポートされていることを確認する必要があります。lgrp_version() 関数には、次の構文があります。

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

lgrp_version() 関数は、lgroup インタフェースのバージョン番号を引数に使用し、システムがサポートしているバージョンを返します。現在の lgroup API の実装で version 引数で指定したバージョン番号がサポートされているときは、lgrp_version() 関数はそのバージョン番号を返します。サポートされていない場合は、lgrp_version() 関数は LGRP_VER_NONE を返します。

例 1–1 lgrp_version() の使用例

```
#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);
    }
```

近傍性グループインタフェースの初期化

アプリケーションは、API を使用して lgroup 階層を参照し、内容を検出するために、lgrp_init(3LGRP) を呼び出す必要があります。lgrp_init() を呼び出すことにより、アプリケーションは lgroup 階層のスナップショットを取得できます。アプリケーション開発者は、特に呼び出したスレッドが利用可能な資源だけをスナップショットに含めるか、あるいはオペレーティングシステム全般で利用可能な資源を含めるかを指定できます。lgrp_init() 関数は cookie を返します。cookie は、次のタスクで使用されます。

- lgroup 階層の参照
- lgroup の内容の判定

- スナップショットが最新であるかどうかの判定

lgrp_init() の使用法

lgrp_init() 関数は、 lgroup インタフェースを初期化し、 lgroup 階層のスナップショットを取得します。

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

引数 view に LGRP_VIEW_CALLER を指定して lgrp_init() 関数を呼び出す場合には、呼び出したスレッドで利用可能な資源だけがスナップショットとして返されます。引数 view に LGRP_VIEW_OS を指定して lgrp_init() 関数を呼び出す場合は、オペレーティングシステムで利用可能な資源がスナップショットとして返されます。スレッドが lgrp_init() 関数の呼び出しに成功すると、関数は lgroup 階層とやりとりするすべての関数で使用される cookie を返します。スレッドに cookie が必要でなくなったときは、cookie を引数に指定して lgrp_fini() 関数を呼び出します。

lgroup 階層は、マシン上のすべての CPU とメモリー資源を含むルート lgroup 階層によって構成されています。ルート lgroup は、応答時間がより短く制限された複数の近傍性グループを持つことができます。

lgrp_init() 関数が返すエラーは 2 種類あります。無効な引数 view が指定された場合、EINVAL を返します。lgroup 階層のスナップショットを割り当てる十分なメモリーがない場合には、ENOMEM を返します。

lgrp_fini() の使用法

lgrp_fini(3LGRP) 関数は、取得した cookie の使用を終了し、対応する lgroup 階層のスナップショットを解放します。

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

lgrp_fini() 関数は、前に lgrp_init() を呼び出して作成した lgroup 階層のスナップショットを表す cookie を引数に使用します。lgrp_fini() 関数は、そのスナップショットに割り当てられたメモリーを解放します。lgrp_fini() を呼び出したあとでは、cookie は無効になります。その cookie を使用することはできません。

lgrp_fini() 関数に渡した cookie が無効な場合、lgrp_fini() は EINVAL を返します。

近傍性グループ階層

この節で説明する API を使用することにより、呼び出しスレッドから lgroup 階層を参照できます。lgroup 階層は、Directed Acyclic Graph (DAG) です。ツリー構造と似ていますが、ノードは複数の親を持つことができます。ルート lgroup はマシン全体を表し、ルート lgroup にはそのマシンのすべての資源が含まれます。ルート lgroup はシステム全体でもっとも高い応答時間値を持っています。それぞれの子 lgroup はルート lgroup にあるハードウェアのサブセットで構成されています。それぞれの子 lgroup の応答時間値はより低く制限されています。ルートに近い近傍性グループほど、多くの資源と高い応答時間が与えられています。葉に近い近傍性グループは、資源も少なく、応答時間も低くなります。lgroup には、その応答時間の範囲内で直下に資源が含まれる場合があります。また、lgroup に葉 lgroup が含まれ、その葉 lgroup に自身の資源セットが含まれる場合もあります。葉 lgroup の資源は、その葉 lgroup をカプセル化する lgroup から利用可能です。

lgrp_cookie_stale() の使用法

lgrp_cookie_stale(3LGRP) 関数は、指定の cookie で表された lgroup 階層のスナップショットが最新のものであるかどうかを判定します。

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

lgrp_init() 関数が返す cookie は、そのスナップショットを取得した view 引数の種類によって、さまざまな理由により無効になる場合があります。view 引数に LGRP_VIEW_OS を指定して呼び出した lgrp_init() 関数が返す cookie では、動的再構成や CPU のオンライン状態の変化などが原因で lgroup 階層に変更があったときに、無効になる場合があります。view 引数に LGRP_VIEW_CALLER を指定した lgrp_init() 関数が返す cookie では、呼び出しスレッドのプロセッサセットの変更または lgroup 階層の変化が原因で、無効になる場合があります。無効になった cookie は、その cookie で lgrp_fini() 関数を呼び出し、次に新規 cookie を生成する lgrp_init() 関数を呼び出すことによって新しくなります。

無効な cookie を指定した場合、lgrp_cookie_stale() 関数は EINVAL を返します。

lgrp_view() の使用法

lgrp_view(3LGRP) 関数は、指定した lgroup 階層のスナップショットがどのビューで取得されたかを判別します。

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

lgrp_view() 関数は、lgroup 階層のスナップショットを表す cookie を引数に使用し、そのスナップショットの view を返します。LGRP_VIEW_CALLER を view に指定して取得したスナップショットには、呼び出しスレッドで使用可能な資源のみが含まれます。LGRP_VIEW_OS で取得したスナップショットには、オペレーティングシステムで使用可能なすべての資源が含まれます。

無効な cookie を指定した場合、lgrp_view() 関数は EINVAL を返します。

lgrp_nlgrps() の使用法

lgrp_nlgrps(3LGRP) 関数は、システムに存在する近傍性グループの数を返します。近傍性グループが 1 つしかないシステムでは、メモリー配置の最適化による効果はありません。

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

lgrp_nlgrps() 関数は、lgroup 階層のスナップショットを表す cookie を引数に使用し、階層で使用可能な lgroup の数を返します。

無効な cookie を指定した場合、lgrp_nlgrps() 関数は EINVAL を返します。

lgrp_root() の使用法

lgrp_root(3LGRP) 関数は、ルート lgroup ID を返します。

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

lgrp_root() 関数は、lgroup 階層のスナップショットを表す cookie を引数に使用し、ルート lgroup ID を返します。

lgrp_parents() の使用法

lgrp_parents(3LGRP) 関数は、lgroup 階層のスナップショットを表す cookie を引数に使用し、指定した lgroup の親 lgroup の数を返します。

```
#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);
```

lgrp_array が NULL ではなく、lgrp_array_size の値がゼロでないとき、lgrp_parents() 関数は、要素数の上限まで親 lgroup ID を配列に入れるか、またはす

べての親 lgroup ID を配列に入れて返します。ルート lgroup には親はありません。ルート lgroup に対して lgrp_parents() 関数が呼び出された場合は、lgrp_array には何も返されません。

無効な cookie を指定した場合、lgrp_parents() 関数は EINVAL を返します。指定した lgroup ID が存在しない場合は、lgrp_parents() 関数は ESRCH を返します。

lgrp_children() の使用法

lgrp_children(3LGRP) 関数は、呼び出しスレッドの lgroup 階層のスナップショットを表す cookie を引数に使用し、指定した lgroup の子 lgroup の数を返します。

```
#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);
```

lgrp_array が NULL ではなく、lgrp_array_size の値がゼロでないとき、lgrp_children() 関数は、要素数の上限まで子 lgroup ID を配列に入れるか、またはすべての子 lgroup ID を配列に入れて返します。

無効な cookie を指定した場合、lgrp_children() 関数は EINVAL を返します。指定した lgroup ID が存在しない場合は、lgrp_children() 関数は ESRCH を返します。

近傍性グループの内容

次に示す API では、指定した lgroup の内容に関する情報を取り出します。

lgroup 階層は、ドメインの資源を組織化し、もっとも近い資源を検索するプロセスを簡素化します。葉 lgroup は、もっとも応答時間の短い資源で定義されます。葉 lgroup の上位ノードの各 lgroup には、その子 lgroup にもっとも近い資源が含まれます。ルート lgroup には、そのドメインにあるすべての資源が含まれます。

lgroup の資源は、lgroup の直下に含まれるか、またはその lgroup にカプセル化された葉 lgroup 内に含まれます。葉 lgroup は、直下に資源を含み、ほかの lgroup をカプセル化することはありません。

lgrp_resources() の使用法

lgrp_resources() 関数は、指定した lgroup に含まれる資源の数を返します。

```
#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);
```

lgrp_resources() 関数は、lgroup 階層のスナップショットを表す cookie を引数に使用します。その cookie は lgrp_init () 関数から取得されます。lgrp_resources() 関数は、lgrp 引数の値で指定した ID を持つ lgroup にある資源の数を返します。lgrp_resources() 関数は、CPU またはメモリーの資源を直下に含む lgroup のセットを持つ資源を表します。lgrp_rsrc_t 引数には、次の 2 つの値が指定できます。

LGRP_RSRC_CPU　lgrp_resources() 関数は CPU 資源の数を返します。

LGRP_RSRC_MEM　lgrp_resources() 関数はメモリー資源の数を返します。

lgrpids[] 引数として渡された値が NULL でなく、かつ count 引数に渡された値がゼロでない場合、lgrp_resources() 関数は lgroup ID を lgrpids[] 配列に格納します。lgrpids[] 配列に格納できる lgroup ID の最大数は count 引数の値です。

指定した cookie、lgroup ID、またはタイプが無効な場合、lgrp_resources() 関数は EINVAL を返します。指定した lgroup ID が見つからない場合、lgrp_resources () 関数は ESRCH を返します。

lgrp_cpus() の使用法

lgrp_cpus(3LGRP) 関数は、lgroup 階層のスナップショットを表す cookie を引数に使用し、指定した lgroup にある CPU の数を返します。

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

cpuid[] 引数が NULL ではなく、CPU 数がゼロでないとき、lgrp_cpus() 関数は、配列の要素数の上限まで、またはすべての CPU ID を配列に入れて返します。

content 引数には、次の 2 つの値が指定できます。

LGRP_CONTENT_ALL　lgrp_cpus() 関数は、この lgroup と下位ノードにある CPU の ID を返します。

LGRP_CONTENT_DIRECT　lgrp_cpus() 関数は、この lgroup にある CPU の ID だけを返します。

指定した cookie、lgroup ID、またはフラグのいずれか 1 つが無効な場合、lgrp_cpus() 関数は EINVAL を返します。指定した lgroup ID が存在しない場合は、lgrp_cpus() 関数は ESRCH を返します。

lgrp_mem_size() の使用法

lgrp_mem_size(3LGRP) 関数は、lgroup 階層のスナップショットを表す cookie を引数に使用し、指定した lgroup にインストールされたメモリー、または空きメモリー領域のサイズを返します。lgrp_mem_size() 関数は、メモリーサイズをバイト数で返します。

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

type 引数には、次の値が指定できます。

LGRP_MEM_SZ_FREE　lgrp_mem_size() 関数は、空きメモリー領域のサイズをバイト数で返します。

LGRP_MEM_SZ_INSTALLED　lgrp_mem_size() 関数は、インストールされたメモリーのサイズをバイト数で返します。

content 引数には、次の 2 つの値が指定できます。

LGRP_CONTENT_ALL　lgrp_mem_size() 関数は、この lgroup と下位ノードのメモリーサイズの総量を返します。

LGRP_CONTENT_DIRECT　lgrp_mem_size() 関数は、この lgroup のメモリーのサイズのみを返します。

指定した cookie、lgroup ID、またはフラグのいずれか 1 つが無効な場合、lgrp_mem_size() 関数は EINVAL を返します。指定した lgroup ID が存在しない場合は、lgrp_mem_size() 関数は ESRCH を返します。

近傍性グループの特性

次に示す API では、指定した lgroup の特性に関する情報を取り出します。

lgrp_latency_cookie() の使用法

lgrp_latency(3LGRP) 関数は、ある lgroup の CPU が別の lgroup のメモリーにアクセスするときの応答時間を返します。

```
#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);
```

lgrp_latency_cookie() 関数は、lgroup 階層のスナップショットを表す cookie を引数に使用します。lgrp_init() 関数がこの cookie を作成します。lgrp_latency_cookie() 関数は、「*from*」引数の値で指定された lgroup にあるハードウェア資源と「*to*」引数の値で指定された lgroup にあるハードウェア資源の間の応答時間を表す値を返します。2 つの引数が同じ lgroup を指している場合 lgrp_latency_cookie() 関数は、同一 lgroup での応答時間値を返します。

注 – lgrp_latency_cookie() 関数が返す応答時間値は、オペレーティングシステムで定義されたもので、プラットフォーム固有の数値です。この値は、実際のハードウェアデバイスの応答時間を表しているとは限りません。あるドメイン内部での比較のためにのみ使用してください。

「*between*」引数の値が LGRP_LAT_CPU_TO_MEM である場合、lgrp_latency_cookie() 関数は CPU 資源からメモリー資源までの応答時間を測定します。

無効な lgroup ID を指定した場合、lgrp_latency_cookie() 関数は EINVAL を返します。lgrp_latency_cookie () 関数で指定された lgroup ID が見つからないとき、「from」引数で指定された lgroup に CPU が存在しないとき、または「to」引数で指定された lgroup にメモリーが存在しないときには、lgrp_latency_cookie() 関数は ESRCH を返します。

近傍性グループ、スレッド、およびメモリー配置

この節では、各 lgroup のスレッドとメモリー配置を検出し、制御するために使用する API について説明します。

- lgrp_home(3LGRP) 関数は、スレッドの配置を検出するために使用します。
- メモリー配置の検出には、meminfo(2) システムコールを使用します。
- lgroup 間でのメモリー配置を制御するには、madvise(3C) 関数に MADV_ACCESS フラグを設定して使用します。
- lgrp_affinity_set(3LGRP) 関数では、指定した lgroup のスレッドのアフィニティーを設定することにより、スレッドとメモリー配置を制御できます。
- ある lgroup のアフィニティーが、資源をどの lgroup から割り当てるか優先順位を決定するのに影響を与える可能性があります。
- カーネルが効率的なメモリーの割り当てを行うには、アプリケーションのメモリー使用パターンについての情報が必要になります。
- madvise() 関数およびその共有オブジェクト版である madv.so.1 により、この情報をカーネルに提供します。

- 実行中のプロセスが meminfo() システムコールを使用して自分自身のメモリー使用に関する情報を収集できます。

lgrp_home() の使用法

lgrp_home() 関数は、指定したプロセスまたはスレッドのホーム lgroup を返します。

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

無効な ID タイプを指定した場合、lgrp_home() 関数は EINVAL を返します。呼び出しプロセスの実効ユーザーがスーパーユーザーではなく、実ユーザー ID または実効ユーザー ID が指定したスレッドの実ユーザー ID または実効ユーザー ID とも一致しない場合、lgrp_home() 関数は EPERM を返します。指定したプロセスまたはスレッドが存在しない場合は、lgrp_home() 関数は ESRCH を返します。

madvise() の使用法

madvise() 関数は、ユーザーの仮想メモリー領域の *addr* で指定された開始アドレスから *len* パラメータの値で示される長さの範囲について特定の使用パターンに従うように、カーネルに対してアドバイスを与えます。カーネルはこの情報を使用して、指定された範囲に関連付けられた資源の操作と管理の手順を最適化します。メモリーに対するアクセスパターンに関する適切な情報を持つプログラムで madvise() 関数を使用できれば、システム性能を向上させることができます。

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

madvise() 関数では、複数 lgroup に対するスレッドのメモリーの割り当て方法を操作するために、次のフラグが提供されています。

MADV_ACCESS_DEFAULT	このフラグは、指定範囲に関して期待されるカーネルのアクセスパターンを取り消してデフォルトに戻します。
MADV_ACCESS_LWP	このフラグは、指定のアドレス範囲に次回アクセスする LWP が、その領域にもっとも頻繁にアクセスする LPW であることをカーネルに指定します。それに応じて、カーネルはメモリーやほかの資源をこの領域と LWP に割り当てます。
MADV_ACCESS_MANY	このフラグは、多くのプロセスまたは LPW が、ランダムにシステム全域から指定の領域にアクセスしていることをカーネルに指定します。それに応じて、カーネルはメモ

リーやほかの資源をこの領域に割り当てます。

madvise() 関数は、次の値を返します。

EAGAIN　*addr* から *addr+len* までのアドレス範囲で指定されたマッピング領域の全体または一部が、入出力操作によりロックされている場合。

EINVAL　*addr* パラメータの値が sysconf(3C) で返されるページサイズの倍数ではない、指定したアドレス範囲の長さがゼロ以下、またはアドバイスが無効の場合。

EIO　ファイルシステムに対する読み取りや書き込み中に入出力エラーが発生した場合。

ENOMEM　指定したアドレス範囲のアドレスが、プロセスの有効なアドレス空間の範囲外、またはマップされていないページが指定されている場合。

ESTALE　NFS ファイルハンドルが無効の場合。

madv.so.1 の使用法

共有オブジェクト madv.so.1 は、起動されたプロセスやその子プロセスに対して選択された仮想メモリーのアドバイスの構成を実現します。共有オブジェクトを使用するには、環境変数に次の文字列を指定する必要があります。

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

madv.so.1 共有オブジェクトは、 MADV 環境変数の値に従ってメモリーのアドバイスを適用します。MADV 環境変数は、プロセスのアドレス空間におけるすべてのヒープ、共有メモリー、および mmap 領域のために使用する仮想メモリーのアドバイスを指定します。この情報は生成されたすべてのプロセスに適用されます。次に示す MADV 環境変数値は、複数の lgroup 間での資源の割り当てに影響を与えます。

access_default　この値は、カーネルに期待されるアクセスパターンをデフォルトに戻します。

access_lwp　この値は、アドレス範囲に次回アクセスする LWP が、その領域にもっとも頻繁にアクセスする LPW であることをカーネルに指定します。それに応じて、カーネルはメモリーやほかの資源をこの領域と LWP に割り当てます。

access_many　この値は、多くのプロセスまたは LPW が、ランダムにシステム全域からメモリーにアクセスしていることをカーネルに指定します。それに応じて、カーネルはメモリーやほかの資源を割り当てます。

MADVCFGFILE 環境変数の値は、1 つまたは複数のメモリーのアドバイス構成エントリが *exec-name*:*advice-opts* の書式で記述されているテキストファイルの名前です。

exec-name の値は、アプリケーションまたは実行プログラムの名前です。*exec-name* には、フルパス名、基本名、またはパターン文字列による指定が可能です。

advice-opts の値は、*region=advice* の書式で表します。*advice* の値は、MADV 環境変数の値と同じです。*region* には、次のいずれかの規定された値を指定します。

madv	プロセスのアドレス空間のすべてのヒープ、共有メモリー、および mmap(2) 領域に、アドバイスが適用されます。
heap	ヒープは、brk(2) 領域として定義されます。アドバイスは、既存のヒープにも将来割り当てられる追加ヒープメモリーにも適用されます。
shm	アドバイスは、共有メモリーセグメントに適用されます。共有メモリー操作に関する詳細は、shmat(2) を参照してください。
ism	アドバイスは、SHM_SHARE_MMU フラグを使用している共有メモリーセグメントに適用されます。ism オプションは、shm より優先されます。
dsm	アドバイスは、SHM_PAGEABLE フラグを使用している共有メモリーセグメントに適用されます。dsm オプションは、shm より優先されます。
mapshared	アドバイスは、MAP_SHARED フラグを使用した mmap() システムコールにより作成されたマッピングに適用されます。
mapprivate	アドバイスは、MAP_PRIVATE フラグを使用した mmap() システムコールにより作成されたマッピングに適用されます。
mapanon	アドバイスは、MAP_ANON フラグを使用した mmap() システムコールにより作成されたマッピングに適用されます。複数のオプションが指定された場合は、mapanon オプションが優先されます。

MADVERRFILE 環境変数値は、エラーメッセージが記録されるパスの名前です。MADVERRFILE による指定がない場合は、madv.so.1 共有オブジェクトは syslog(3C) を使用してエラーを記録します。重要度は LOG_ERR、機能記述子は LOG_USER になります。

メモリーに関するアドバイスは継承されます。子プロセスには親と同じアドバイスが適用されます。madv.so.1 共有オブジェクトにより異なるレベルのアドバイスを設定しないかぎり、exec(2) が呼び出されたあとには、アドバイスはシステムデフォルトの設定に戻されます。アドバイスは、ユーザープログラムによって作成された mmap() 領域にのみ適用されます。実行時リンカーまたはシステムライブラリによって作成された直接システムコールを呼び出す領域は影響を受けません。

madv.so.1の使用例

次の例では、madv.so.1共有オブジェクトの機能について個別に説明します。

例1–2 アプリケーションセットへのアドバイスの設定

この設定では、execがfooで始まるすべてのアプリケーションのISMセグメントにアドバイスを適用しています。

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

例1–3 特定のアプリケーションセットに対するアドバイスの除外

この設定では、lsを除くすべてのアプリケーションにアドバイスを適用しています。

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

例1–4 構成ファイルでのパターンマッチの使用

MADVCFGFILEに指定された構成はMADVの設定値より優先されるので、ファイルの最後の構成エントリで*exec-name*に指定した*は、MADVを指定した場合と同じ意味になります。この例は、前述の例と同じ結果になります。

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

例1–5 複数の領域に対するアドバイス

この構成では、あるアドバイスのタイプをmmap()領域に適用し、さらに別のアドバイスのタイプをexec()の名前がfooで始まるアプリケーションのヒープおよび共有メモリー領域に対して適用しています。

例 1–5　複数の領域に対するアドバイス　*(続き)*

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

meminfo() の使用法

meminfo() 関数は、システムにより割り当てられた仮想メモリーおよび物理メモリーに関する情報を、呼び出しプロセスに提供します。

```
#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[]);
```

meminfo() 関数は、次に示す情報を返します。

MEMINFO_VPHYSICAL	指定した仮想アドレスに対応する物理メモリーのアドレス
MEMINFO_VLGRP	指定した仮想アドレスに対応する物理ページがある lgroup
MEMINFO_VPAGESIZE	指定した仮想アドレスに対応する物理ページのサイズ
MEMINFO_VREPLCNT	指定した仮想アドレスに対応する物理ページの複製の数
MEMINFO_VREPL|n	指定した仮想アドレスの *n* 番目の物理ページの複製
MEMINFO_VREPL_LGRP|n	指定した仮想アドレスの *n* 番目の物理ページの複製がある lgroup
MEMINFO_PLGRP	指定した物理アドレスがある lgroup

meminfo() 関数には、次のパラメータを指定できます。

inaddr	入力アドレスの配列。
addr_count	meminfo() 関数に渡されるアドレスの数。
info_req	要求される情報のタイプをリストする配列。
info_count	*inaddr* 配列にある各アドレスについて要求された情報の数。
outdata	meminfo() 関数が結果を返す配列。配列のサイズは、*info_req* と *addr_count* パラメータに指定した値の積になります。

validity *addr_count* パラメータの値と同じサイズの配列。*validity* 配列にはビット単位の結果コードが返されます。結果コードの *0* 番目のビットでは、対応する入力アドレスの有効性が評価されています。続くビットでは、*info_req* 配列のメンバーへの応答の有効性が順番に評価されています。

outdata または *validity* 配列が指しているメモリー領域に書き込めない場合、`meminfo()` 関数は `EFAULT` を返します。*info_req* または *inaddr* 配列が指しているメモリー領域から読み込めない場合は、`meminfo()` 関数は `EFAULT` を返します。*info_count* の値が 31 より大きいか、または 1 より小さい場合は、`meminfo()` 関数は `EINVAL` を返します。*addr_count* の値がゼロより小さい場合は、`meminfo()` 関数は `EINVAL` を返します。

例 1–6 仮想アドレスのセットに対応する物理ページとページサイズを出力する `meminfo()` の使用法

```
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]);
```

例 1–6 仮想アドレスのセットに対応する物理ページとページサイズを出力する meminfo() の使用法 (続き)

```
                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);
                }
        }
}
```

近傍性グループのアフィニティー

スレッドの軽量プロセス (LWP) が作成されるとき、カーネルはスレッドに近傍性グループを割り当てます。そのような lgroup は、スレッドの「ホーム lgroup」と呼ばれます。カーネルは、スレッドをホーム lgroup の CPU で実行し、可能な限り lgroup からメモリーを割り当てます。ホーム lgroup の資源が使用可能でない場合は、ほかの lgroup から資源を割り当てます。スレッドに 1 つまたは複数の lgroup に対するアフィニティーがある場合は、オペレーティングシステムはアフィニティーの強さに応じて lgroup から順に資源を割り当てます。lgroup は、次の 3 つのアフィニティーレベルのうち 1 つを持つことができます。

1. LGRP_AFF_STRONG は強いアフィニティーを示します。この lgroup がスレッドのホーム lgroup であるとき、オペレーティングシステムは、そのスレッドのホーム lgroup を変更する再ホーミングをできるだけ避けようとします。その場合でも、動的再構成、プロセッサのオフライン化、プロセッサ割り当て、プロセッサセットの割り当ておよび操作などのイベントは、スレッドの再ホーミングにつながる可能性があります。
2. LGRP_AFF_WEAK は、弱いアフィニティーを示します。この lgroup がスレッドのホーム lgroup であるとき、オペレーティングシステムは、負荷均衡の必要に応じてスレッドの再ホーミングを行います。
3. LGRP_AFF_NONE はアフィニティーを持たないことを示します。スレッドがどの lgroup にもアフィニティーを持たないとき、オペレーティングシステムはそのスレッドにホーム lgroup を割り当てます。

指定されたスレッドに資源を割り当てるときに、オペレーティングシステムは lgroup のアフィニティーをアドバイスとして使用します。このアドバイスはほかのシステム制限とともに考慮されます。プロセッサ割り当ておよびプロセッサセット

によってlgroupのアフィニティーが影響を受けることはありませんが、スレッドが実行されるlgroupを制限する場合があります。

lgrp_affinity_get()の使用法

lgrp_affinity_get(3LGRP)関数は、指定したlgroupに対してLWPが持つアフィニティーを返します。

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

*idtype*および*id*引数により、lgrp_affinity_get()関数で検証するLWPを指定します。*idtype*の値にP_PIDを指定した場合、lgrp_affinity_get()関数は、*id*引数の値と一致するプロセスIDを持つプロセスにあるLWPのいずれかについてlgroupアフィニティーを取得します。*idtype*にP_LWPIDを指定した場合、lgrp_affinity_get()関数は、実行中のプロセスにある、*id*引数の値と一致するLWP IDを持つLWPについてlgroupアフィニティーを取得します。*idtype*にP_MYIDを指定した場合、lgrp_affinity_get()関数は、実行中のLPWについてlgroupアフィニティーを取得します。

指定したlgroupまたはIDタイプが有効でないとき、lgrp_affinity_get()関数はEINVALを返します。呼び出しプロセスの実効ユーザーがスーパーユーザーではなく、呼び出しプロセスのIDがどのLPWの実ユーザーIDまたは実効ユーザーIDとも一致しない場合、lgrp_affinity_get()関数はEPERMを返します。指定したlgroupまたはLPWが存在しない場合は、lgrp_affinity_get()関数はESRCHを返します。

lgrp_affinity_set()の使用法

lgrp_affinity_set(3LGRP)関数は、指定したlgroupに対してLWPまたはLWPのセットが持つアフィニティーを設定します。

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

*idtype*および*id*引数により、lgrp_affinity_set()関数で検証するLWPまたはLWPのセットを指定します。*idtype*にP_PIDを指定した場合、lgrp_affinity_set()関数は、*id*引数の値と一致するプロセスIDを持つプロセスのすべてのLWPについて、lgroupアフィニティーを*affinity*引数で指定したアフィニティーレベルに設定します。*idtype*にP_LWPIDを指定した場合、lgrp_affinity_set()関数は、*id*引数の値と一致するLWP IDを持つ実行プロセス中のLWPについて、lgroupアフィニティーを*affinity*引数で指定したアフィニティーレベルに設定します。*idtype*にP_MYIDを指定した場合は、lgrp_affinity_set()関数は、実行中のLWPまたはプロセスについて、lgroupアフィニティーを*affinity*引数で指定したアフィニティーレベルに設定します。

指定したlgroup、アフィニティー、またはIDタイプが有効でないとき、`lgrp_affinity_set()`関数は`EINVAL`を返します。呼び出しプロセスの実効ユーザーがスーパーユーザーではなく、呼び出しプロセスのIDがどのLPWの実ユーザーIDまたは実効ユーザーIDとも一致しない場合、`lgrp_affinity_set()`関数は`EPERM`を返します。指定したlgroupまたはLPWが存在しない場合は、`lgrp_affinity_set()`関数は`ESRCH`を返します。

APIの使用例

この節では、この章で説明したAPIを使用するタスクのコーディング例を示します。

例1–7 スレッドへのメモリーの移動

次のコーディング例では、*addr*と*addr+len*のアドレス範囲にあるメモリーを、その範囲に次回アクセスするスレッド付近に移動します。

```
#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");
}
```

例1–8 メモリーへのスレッドの移動

このコーディング例では、`meminfo()`関数を使用して、指定されたアドレスで仮想ページにバックアップする物理メモリーのlgroupを判定します。次にこの例では、現在のスレッドをそのメモリー付近に移動するために、そのlgroupに強いアフィニティーを設定します。

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

例1–8 メモリーへのスレッドの移動 (続き)

```
/*
 * 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);
}
```

例1–9 lgroup階層の巡回

次のコーディング例では、lgroup階層を巡回し、出力します。

例 1–9 lgroup階層の巡回 (続き)

```
#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) {
```

例 1–9 lgroup階層の巡回 *(続き)*

```
        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);
```

例 1–9 lgroup階層の巡回 *(続き)*

```
                }
        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);
}
```

例 1–10 指定されたlgroup以外で利用可能なメモリーを持つもっとも近いlgroupの検索

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

例 1–10 指定されたlgroup以外で利用可能なメモリーを持つもっとも近いlgroupの検索 *(続き)*

```
#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);
```

例 1–10 指定されたlgroup以外で利用可能なメモリーを持つもっとも近いlgroupの検索 (続き)

```
        }

    /*
     * 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;
```

例 1–10 指定されたlgroup以外で利用可能なメモリーを持つもっとも近いlgroupの検索 *(続き)*

```
            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));
}
```

例1–11 空きメモリー領域を持つもっとも近いlgroupの検索

このコーディング例では、指定されたスレッドのホームlgroupにもっとも近い、空きメモリー領域を持つlgroupを検索します。

```
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));
}
```

◆ ◆ ◆ 第 2 章

MPO 可観測性ツール

この章では、Solaris オペレーティングシステムに用意されている MPO 機能を使用する場合に利用できるツールについて説明します。

次の内容について説明します。

- 37 ページの「pmadvise ユーティリティー」では、プロセスでのメモリーの使用方法を定義する規則を適用するためのツールについて説明します。
- 39 ページの「plgrp ツール」では、近傍性グループに対するスレッドのアフィニティーを表示および設定するためのツールについて説明します。
- 41 ページの「lgrpinfo ツール」では、lgroup の階層、内容、および特性に関する情報を出力するためのツールについて説明します。
- 43 ページの「Solaris::lgrp モジュール」では、第 1 章で説明されている近傍性グループ API への Perl インタフェースについて説明します。

pmadvise ユーティリティー

pmadvise ユーティリティーは、プロセスでのメモリーの使用方法を定義する規則をプロセスに適用します。pmadvise ユーティリティーは、madvise(3C) ツールを使用して、「アドバイス」と呼ばれる規則をプロセスに適用します。このツールは、特定の時間に、メモリー位置の特定の部分範囲に対してアドバイスを適用することができます。一方、madv.so.1(1) ツールは、ターゲットプログラムの実行期間全体にわたり、指定されたタイプのセグメントすべてに対してアドバイスを適用します。

pmadvise ユーティリティーには次のオプションがあります。

-f　このオプションは、ターゲットプロセスを制御下に置きます。このオプションは、ほかのどのプロセスの制御よりも優先します。proc(1) のマニュアルページを参照してください。

-o　このオプションは、ターゲットプロセスに適用するアドバイスを指定します。アドバイスは、次の形式で指定してください。

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

advice の値には、次のいずれかを指定できます。

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

アドレスと長さを指定して、アドバイスを適用する部分範囲を指定することができます。アドレスは 16 進表記、長さはバイト数で指定してください。

長さが指定されていない場合で、開始アドレスがセグメントの先頭を指しているときは、pmadvise ユーティリティーはそのセグメントにアドバイスを適用します。長さには修飾文字 K、M、G、T、P、または E を追加できます。これらはそれぞれ、キロバイト、メガバイト、ギガバイト、テラバイト、ペタバイト、およびエクサバイトを表します。

`-v` このオプションは、現在適用されているアドバイス規則の値および位置を、`pmap(1)` ツールの書式の詳細出力で表示します。

pmadvise ツールは、正当なオプションはすべて処理しようとします。オプションで不正なアドレス範囲が指定されている場合、pmadvise ツールはそのオプションの処理を試みるときにエラーメッセージを表示し、そのオプションをスキップします。構文エラーを検出した場合、pmadvise ツールはどのオプションも処理せずに終了し、使用法のメッセージを表示します。

特定の領域に対するアドバイスが、より一般的な領域に対するアドバイスと衝突している場合は、特定の領域に対するアドバイスが優先されます。特定のアドレス範囲を指定するアドバイスは、ヒープ領域およびスタック領域に対するアドバイスより優先されます。また、ヒープ領域およびスタック領域に対するアドバイスは、プライベートメモリーおよび共有メモリーに対するアドバイスより優先されます。

次の各グループに含まれるアドバイス規則は、同じグループ内のほかのアドバイス規則とは互いに排他的です。

```
MADV_NORMAL, MADV_RANDOM, MADV_SEQUENTIAL
MADV_WILLNEED, MADV_DONTNEED, MADV_FREE
```

MADV_ACCESS_DEFAULT, MADV_ACCESS_LWP, MADV_ACCESS_MANY

plgrp ツール

plgrp ユーティリティーは、1 つ以上のプロセス、スレッド、または軽量プロセス (LWP) について、ホーム lgroup と lgroup のアフィニティーを表示したり設定したりできます。各スレッドには、作成時にホーム lgroup が割り当てられます。CPU 資源またはメモリー資源をスレッドに割り当てるとき、システムは、スレッドのホームにもっとも近い位置にある使用可能な資源を探すために、スレッドのホーム lgroup から順番に lgroup 階層を検索します。

システムにより、各スレッドのホーム lgroup が選択されます。ホーム lgroup に対するスレッドのアフィニティーは、最初は none、つまりアフィニティーなしに設定されます。スレッドが、ホーム lgroup に対するスレッドのアフィニティーより高いアフィニティーを、プロセッサセット内の lgroup に対して設定すると、システムはスレッドをその lgroup に移動します。CPU に割り当てられているスレッドは移動されません。ホーム lgroup に対するスレッドのアフィニティーが削除 (none に設定) されると、プロセッサセット内でもっとも高いアフィニティーを持っている lgroup がホーム lgroup としてスレッドに割り当て直されます。

lgroup のアフィニティーのさまざまなレベルおよびその意味の詳細については、lgrp_affinity_set(3LGRP) のマニュアルページを参照してください。

plgrp ツールでは、次のオプションがサポートされています。

-a *lgroup list*	このオプションは、リスト内の lgroup に対して、指定されたプロセスまたはスレッドのアフィニティーを表示します。
-A*lgroup list*/ none|weak|strong[,...]	このオプションは、リスト内の lgroup に対して、指定されたプロセスまたはスレッドのアフィニティーを設定します。*lgroup*/*affinity* の組み合わせをコンマで区切って並べたリストを使用して、複数のアフィニティーを一度に設定することができます。
-F	このオプションは、ターゲットプロセスを制御下に置きます。このオプションは、ほかのどのプロセスの制御よりも優先します。proc(1) のマニュアルページを参照してください。
-h	このオプションは、指定されたプロセスまたはスレッドのホーム lgroup を返します。

	これは、オプションを指定せずに plgrp ツールを使用した場合のデフォルトの動作です。
-H *lgroup list*	このオプションは、指定されたプロセスまたはスレッドのホーム lgroup を設定します。このオプションは、リスト内の lgroup に対して強いアフィニティーを設定します。plgrp ユーティリティーは、複数の lgroup が指定された場合、ラウンドロビン方式でこれらの lgroup をホーム lgroup としてスレッドに割り当てようとします。

lgroup の指定

lgroup list 変数の値は、次の属性の 1 つ以上をコンマで区切って並べたリストです。

- lgroup ID
- *start lgroup ID-end lgroup ID* という形式で指定する、lgroup ID の範囲
- `all`
- `root`
- `leaves`

`all` キーワードは、システム内のすべての lgroup ID を表します。`root` キーワードは、ルート lgroup の ID を表します。`leaves` キーワードは、すべての葉 lgroup の ID を表します。葉 lgroup とは、子を持っていない lgroup のことです。

プロセスおよびスレッドの引数の指定

plgrp ユーティリティーは、1 つ以上のプロセスまたはスレッドを空白文字で区切って並べたリストを引数として受け取ります。proc(1) ツールで使用される構文と同じ構文で、プロセスとスレッドを指定できます。*pid* または /proc/*pid* という構文を使用して、プロセス ID を整数で指定できます。/proc/*pid* 構文ではシェル展開を使用できます。プロセス ID だけを指定すると、plgrp ユーティリティーの引数には、そのプロセスのすべてのスレッドが含められます。

pid/*lwpid* という構文でプロセス ID とスレッド ID を指定することにより、スレッドを明示的に指定できます。「-」文字を使用して一度に範囲を選択するか、コンマで区切って並べたリストを使用して、プロセス内の複数のスレッドを指定できます。*pid* というプロセス ID を持つプロセスのスレッド 1、2、7、8、および 9 を指定するには、構文 *pid*/1,2,7-9 を使用します。

lgrpinfo ツール

lgrpinfo ツールは、lgroup の階層、内容、および特性に関する情報を出力します。lgrpinfo ツールは、Solaris::Lgrp モジュールを必要とする Perl スクリプトです。このツールは、liblgrp(3LIB) API を使用してシステムから情報を取得し、人が読める形式で出力します。

引数を指定せずに lgrpinfo ツールを呼び出すと、システム内のすべての lgroup に関する一般的な情報が出力されます。コマンド行で lgrpinfo ツールに lgroup ID を渡すと、指定した lgroup に関する情報が返されます。lgroup を指定するには、その lgroup ID を使用するか、次のキーワードのいずれかを使用することができます。

all	このキーワードは、すべての lgroup を指定します。これはデフォルトの動作です。
root	このキーワードは、ルート lgroup を指定します。
leaves	このキーワードは、すべての葉 lgroup を指定します。葉 lgroup とは、lgroup 階層内で子を持っていない lgroup のことです。
intermediate	このキーワードは、すべての中間 lgroup を指定します。中間 lgroup とは、親と子を持っている lgroup のことです。

lgrpinfo ツールは、無効な lgroup ID を受け取った場合、その無効な ID を示すメッセージを表示し、コマンド行で渡されたほかの lgroup の処理を続行します。lgrpinfo ツールは、引数内に無効な lgroup がない場合、状態 2 で終了します。

lgrpinfo ツールのオプション

引数を指定せずに lgrpinfo ツールを呼び出すと、オプション -celmrt all を指定した場合と同じ動作になります。lgrpinfo ツールの有効なオプションは次のとおりです。

-a	このオプションは、指定された lgroup ID について、トポロジ、CPU、メモリー、負荷、および応答時間値の情報を出力します。このオプションを指定すると、-T オプションも指定した場合を除き、-tcemrlL オプションの動作を組み合わせた動作になります。-T オプションを指定した場合は、-a オプションの動作に -t オプションの動作は含まれなくなります。
-c	このオプションは、CPU の情報を出力します。
-C	このオプションは、リスト内の各 lgroup をその子で置き換えます。このオプションを -P オプションまたは -T オプションと組み合わせることはできません。引数がまったく指定されていない場合、このオプションはすべての lgroup に適用されます。
-e	このオプションは、葉 lgroup に関する lgroup 平均負荷率を出力します。

`-G` このオプションは、lgroup 階層を OS 表示で出力します。ツールのデフォルトの動作では、lgroup 階層は呼び出し元の表示で出力されます。呼び出し元の表示には、呼び出し元で使用できる資源だけが含まれています。OS 表示と呼び出し元表示の詳細については、`lgrp_init(3LGRP)` のマニュアルページを参照してください。

`-h` このオプションは、ツールに関するヘルプメッセージを表示します。

`-I` このオプションは、指定された ID と一致する ID だけを出力します。このオプションは、`-c`、`-G`、`-C`、または `-P` オプションと組み合わせることができます。`-c` オプションを指定すると、一致するすべての lgroup に含まれている CPU のリストが出力されます。`-c` オプションを指定しない場合は、一致する lgroup の ID が表示されます。引数がまったく指定されていない場合、このオプションはすべての lgroup に適用されます。

`-l` このオプションは、lgroup の応答時間に関する情報を出力します。各 lgroup について示される応答時間値は、オペレーティングシステムによって定義されたもので、プラットフォーム固有の数値です。実行中のシステムで lgroup を相対的に比較する目的にのみ使用できます。この値は、ハードウェアデバイス間の実際の応答時間を表しているとは限らず、異なるプラットフォームには当てはまらない場合もあります。

`-L` このオプションは、lgroup の応答時間表を出力します。この表は、各 lgroup からほかの各 lgroup に対する相対応答時間を示します。

`-m` このオプションは、メモリーの情報を出力します。メモリーサイズの報告には、サイズが 0 から 1023 の整数範囲に収まるような単位が使用されます。`-u` オプションを使用すると、この動作を変更することができます。値が 10 より小さい場合のみ、小数部も表示されます。

`-P` このオプションは、リスト内の各 lgroup をその親で置き換えます。このオプションを `-C` オプションまたは `-T` オプションと組み合わせることはできません。引数がまったく指定されていない場合、このオプションはすべての lgroup に適用されます。

`-r` このオプションは、lgroup の資源に関する情報を出力します。`-T` オプションを指定すると、中間 lgroup の資源についてのみ情報が表示されます。

`-t` このオプションは、lgroup のトポロジに関する情報を出力します。

`-T` このオプションは、システムの lgroup トポロジをツリー形式でグラフィカルに表示します。このオプションは、`-a`、`-c`、`-e`、`-G`、`-l`、`-L`、`-m`、`-r`、および `-u` オプションとの組み合わせでのみ使用できます。出力を中間 lgroup だけに制限するには、`-r` オプションを使用します。`-T` オプションと `-a` オプションを組み合わせるときは、`-t` オプションは省略します。ルート lgroup が唯一の lgroup である場合を除き、このオプションではルート lgroup に関する情報は出力されません。

-u*units* このオプションは、メモリーの単位を指定します。*units* 引数の値には、b、k、m、g、t、p、または e を指定できます。これらはそれぞれ、バイト、キロバイト、メガバイト、ギガバイト、テラバイト、ペタバイト、およびエクサバイトを表します。

Solaris::lgrp モジュール

この Perl モジュールは、liblgrp に含まれている lgroup API への Perl インタフェースを提供します。このインタフェースを使用すると、lgroup 階層を探索し、その内容と特性を検出し、lgroup に対するスレッドのアフィニティーを設定することができます。このモジュールを使用して、lgrp_user.h ヘッダーファイルで定義されているさまざまな定数や関数にアクセスできます。このモジュールには、ライブラリへの手続き型インタフェースとオブジェクトインタフェースが用意されています。

このモジュールは、デフォルトの動作では何もエクスポートしません。次のタグを使用すると、このモジュールで定義されている定数と関数を選択的にインポートできます。

```
: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()
```

この Perl モジュールには、次のメソッドがあります。

- new()
- cookie()
- stale()
- view()
- root()

- children()
- parents()
- nlgrps()
- mem_size()
- cpus()
- isleaf()
- resources()
- version()
- home()
- affinity_get()
- affinity_set()
- lgrps()
- leaves()
- latency()

:CONSTANTS タグまたは :ALL タグを使用して、定数をエクスポートできます。次に示す定数はどれでも、Perl プログラムで使用することができます。

- 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

基になるライブラリ関数が失敗すると、このモジュール内の関数は undef または空のリストを返します。モジュールでは、次のエラーコードが使用されます。

EINVAL　指定された値が無効です。

ENOMEM　システムメモリーが不足したため操作を完了できませんでした。

ESRCH　指定されたプロセスまたはスレッドが見つかりませんでした。

EPERM　呼び出しプロセスの実効ユーザーが適切な特権を持っておらず、実ユーザー ID または実効ユーザー ID が特定のスレッドの実ユーザー ID または実効ユーザー ID とも一致していません。