SunOS リファレンスマニュアル 4: ファイル形式

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

Part No: 819–1216–12
2006 年 5 月

原典: man pages section 4: File Formats

Part No: 816-5174-10

Revision A

060712@14986

目次

はじめに

概要

SunOS リファレンスマニュアルは、初めて SunOS を使用するユーザーやすでにある程度の知識を持っているユーザーのどちらでも対応できるように解説されています。このマニュアルを構成するマニュアルページは一般に参照マニュアルとして作られており、チュートリアルな要素は含んでいません。それぞれのコマンドを実行すると、どのような結果が得られるかについて、詳しく説明されています。なお、各マニュアルページの内容はオンラインでも参照することができます。

このマニュアルは、マニュアルページの内容によっていくつかのセクションに分かれています。各セクションについて以下に簡単に説明します。

- セクション 1 は、オペレーティングシステムで使えるコマンドを説明します。
- セクション 1M は、システム保守や管理用として主に使われるコマンドを説明します。
- セクション 2 は、すべてのシステムコールについて説明します。ほとんどのシステムコールに 1 つまたは複数のエラーがあります。エラーの場合、通常ありえない戻り値が返されます。
- セクション 3 は、さまざまなライブラリ中の関数について説明します。ただし、UNIX システムプリミティブを直接呼び出す関数については、セクション 2 で説明しています。
- セクション 4 は、各種ファイルの形式について説明します。また、ファイル形式を宣言する C 構造体を適用できる場合には随時説明しています。
- セクション 5 は、文字セットテーブルなど他のセクションには該当しないものについて説明します。
- セクション 7 は、特殊なハードウェア周辺装置またはデバイスドライバに関するさまざまな特殊ファイルについて説明します。STREAMS ソフトウェアドライバ、モジュール、またはシステムコールの STREAMS 汎用セットについても説明します。
- セクション 9 は、カーネル環境でデバイスドライバを記述するのに必要な参照情報を提供します。ここでは、デバイスドライバインタフェース (DDI) とドライバ/カーネルインタフェース (DKI) という 2 つのデバイスドライバインタフェース仕様について説明します。
- セクション 9F は、デバイスドライバが使用できるカーネル関数について説明します。

以下に、このマニュアルの項目を表記されている順に説明します。ほとんどのマニュアルページが下記の項目からなる共通の書式で書かれていますが、必要でない項目については省略されています。たとえば、記述すべきバグがコマンドにない場合などは、「使用上の留意点」という項目はありません。各マニュアルページの詳細は各セクションの `intro` を、マニュアルページの一般的な情報については `man`(1) を参照してください。

名前　コマンドや関数の名称と概略が示されています。

形式　コマンドや関数の構文が示されています。標準パスにコマンドやファイルが存在しない場合は、フルパス名が示されます。字体は、コマンド、オプションなどの定数にはボールド体 (bold) を、引数、パラメータ、置換文字などの変数にはイタリック体 (Italic) または <日本語訳> を使用しています。オプションと引数の順番は、アルファベット順です。特別な指定が必要な場合を除いて、1 文字の引数、引数のついたオプションの順に書かれています。

以下の文字がそれぞれの項目で使われています。

- [ ]　このかっこに囲まれたオプションや引数は省略できます。このかっこが付いていない場合には、引数を必ず指定する必要があります。
- . . .　省略符号。前の引数に変数を付けたり、引数を複数指定したりできることを意味します (例: ‘`filename` . . .’)。
- |　区切り文字 (セパレータ)。この文字で分割されている引数のうち 1 つだけを指定できます。
- { }　この大かっこに囲まれた複数のオプションや引数は省略できます。かっこ内を 1 組として扱います。

プロトコル　この項が使われているのは、プロトコルが記述されているファイルを示すサブセクション 3R だけです。パス名は常にボールド体 (bold) で示されています。

機能説明　コマンドの機能とその動作について説明します。実行時の詳細を説明していますが、オプションの説明や使用例はここでは示されていません。対話形式のコマンド、サブコマンド、リクエスト、マクロ、関数などに関しては「使用法」で説明します。

IOCTL　セクション 7 だけに使用される項です。`ioctl`(2) システムコールへのパラメータは `ioctl` と呼ばれ、適切なパラメータを持つデバイスクラスのマニュアルページだけに記載されています。特定のデバイスに関する `ioctl` は、(そのデバイスのマニュアルページに) アルファベット順に記述されています。デバイスの特定のクラスに関する `ioctl` は、`mtio`(7I) のように `io` で終わる名前が付いているデバイスクラスのマニュアルページに記載されています。

オプション	各オプションがどのように実行されるかを説明しています。「形式」で示されている順に記述されています。オプションの引数はこの項目で説明され、必要な場合はデフォルト値を示します。
オペランド	コマンドのオペランドを一覧表示し、各オペランドがコマンドの動作にどのように影響を及ぼすかを説明しています。
出力	コマンドによって生成される出力 (標準出力、標準エラー、または出力ファイル) を説明しています。
戻り値	値を返す関数の場合、その値を示し、値が返される時の条件を説明しています。関数が 0 や -1 のような一定の値だけを返す場合は、値と説明の形で示され、その他の場合は各関数の戻り値について簡単に説明しています。void として宣言された関数はこの項では扱いません。
エラー	失敗の場合、ほとんどの関数はその理由を示すエラーコードを `errno` 変数の中に設定します。この項ではエラーコードをアルファベット順に記述し、各エラーの原因となる条件について説明します。同じエラーの原因となる条件が複数ある場合は、エラーコードの下にそれぞれの条件を別々のパラグラフで説明しています。
使用法	この項では、使用する際の手がかりとなる説明が示されています。特定の決まりや機能、詳しい説明の必要なコマンドなどが示されています。組み込み機能については、以下の小項目で説明しています。 コマンド 修飾子 変数 式 入力文法
使用例	コマンドや関数の使用例または使用方法を説明しています。できるだけ実際に入力するコマンド行とスクリーンに表示される内容を例にしています。例の中には必ず `example%` のプロンプトが出てきます。スーパーユーザーの場合は `example#` のプロンプトになります。例では、その説明、変数置換の方法、戻り値が示され、それらのほとんどが「形式」、「機能説明」、「オプション」、「使用法」の項からの実例となっています。
環境	コマンドや関数が影響を与える環境変数を記述し、その影響について簡単に説明しています。
終了ステータス	コマンドが呼び出しプログラムまたはシェルに返す値と、その状態を説明しています。通常、正常終了には 0 が返され、0 以外の値はそれぞれのエラー状態を示します。

ファイル	マニュアルページが参照するファイル、関連ファイル、およびコマンドが作成または必要とするファイルを示し、各ファイルについて簡単に説明しています。
属性	属性タイプとその対応する値を定義することにより、コマンド、ユーティリティ、およびデバイスドライバの特性を一覧しています。詳細は `attributes`(5) を参照してください。
関連項目	関連するマニュアルページ、当社のマニュアル、および一般の出版物が示されています。
診断	エラーの発生状況と診断メッセージが示されています。メッセージはボールド体 (bold) で、変数はイタリック体 (Italic) または <日本語訳> で示されており、C ロケール時の表示形式です。
警告	作業に支障を与えるような現象について説明しています。診断メッセージではありません。
注意事項	それぞれの項に該当しない追加情報が示されています。マニュアルページの内容とは直接関係のない事柄も参照用に扱っています。ここでは重要な情報については説明していません。
使用上の留意点	すでに発見されているバグについて説明しています。可能な場合は対処法も示しています。

参 照

序章

名前

Intro – ファイル形式の序章

機能説明

本セクションでは、さまざまなファイルの形式についての概要を説明します。C 言語の構造体によるファイル形式の宣言も適宜示します。通常、C の構造体宣言を含むヘッダーは、ディレクトリ `/usr/include` または `/usr/include/sys` にあります。ただし、C プログラムへインクルードするには、`#include <`*filename.h*`>` または `#include <sys/`*filename.h*`>` という構文を使用する必要があります。

ファイル形式一覧

名前	説明
Intro(4)	ファイル形式の序章
power.conf(4)	電源管理システムの設定情報ファイル

参 照

ファイル形式

名前　contract – 契約ファイルシステム

形式　/system/contract

機能説明　/system/contract ファイルシステムは、契約サブシステムへの第一インタフェースとして機能します。/system/contract のサブディレクトリは、利用可能な契約タイプごとに 1 つずつ存在しています。

/system/contract は、標準の /system/contract マウントポイントに加え、任意のマウントポイントにもマウントできます。また、いくつかの場所に同時にマウントすることもできます。そのような追加マウントが許可されているのは、chroot(1M) を使ってファイルシステムのサブツリーにプロセスを制限しやすくしつつも、そうしたプロセスが引き続き、contract コマンドやインタフェースを使用できるようにするためです。

/system/contract のファイルにアクセスするには、標準のシステムコール (open(2)、close(2)、poll(2) など) と libcontract(3LIB) への呼び出しを組み合わせて使用します。

契約ファイルシステムのコンシューマは、大規模ファイルに対応している必要があります。largefile(5) および lfcompile64(5) を参照してください。

ディレクトリ構造　/system/contract ディレクトリの最上位レベルには、利用可能な契約タイプごとに用意されたタイプ固有の名前を持ついくつかのサブディレクトリと、1 つの特殊ディレクトリ all が含まれています。これらのディレクトリはそれぞれ、誰でも読み取りや検索を行えます。

/system/contract/*type* の構造　各 /system/contract/*type* ディレクトリには、ある固定数のファイルが含まれます。また、個数の固定されていないサブディレクトリも含まれます。それらのサブディレクトリは、タイプ *type* の既存の契約に対応したものであり、それらの契約の ID の 10 進表現に基づいた名前を持っています。

/system/contract/*type* ディレクトリ内に存在するファイルは、次のとおりです。

template　このファイルをオープンすると、新しい *type* 契約テンプレートに対するファイル記述子が返されます。

テンプレートファイル記述子では、次の libcontract(3LIB) 呼び出しを使用できます。

ct_tmpl_activate(3contract)
ct_tmpl_clear(3contract)
ct_tmpl_create(3contract)

その他のテンプレート関数については、「条項」を参照してください。

latest　このファイルをオープンすると、オープン元の LWP が最後に書き込んだ *type* 契約の状態ファイルに対するファイル記述子が返されます。「/system/contract/*type*/*id* の構造」を参照してください。

オープン元の LWP が *type* 契約をまだ 1 つも作成していなかった場合、最新契約オープン処理は ESRCH で失敗します。

bundle
: このファイルをオープンすると、システム上のすべての *type* 契約からのイベントを受信するイベントエンドポイントに対するファイル記述子が返されます。タイプバンドルイベントエンドポイントは、特権がなくてもオープンできます。読み取り元の実効ユーザー ID 以外のユーザーが所有し、書き込みを行った契約から送信されてきたイベントは、不可視になります。つまり、それらのイベントは自動的にスキップされます。ただし、その読み取り元の実効セットに {PRIV_CONTRACT_OBSERVER} が含まれている場合はその限りではありません。「イベント」を参照してください。

pbundle
: このファイルをオープンすると、オープン元のプロセスが保持するすべての *type* 契約からのイベントを受信するイベントエンドポイントに対するファイル記述子が返されます。「イベント」を参照してください。

/system/contract/all の構造

/system/contract/all ディレクトリには、数値に基づく名前が付けられたファイルが、システム内の契約ごとに 1 つずつ含まれています。各ファイルは、対応する契約のタイプ固有ディレクトリへのシンボリックリンクになっています。つまり、/system/contract/all/*id* は、/system/contract/*type*/*id* をポイントしています。

/system/contract/*type*/*id* の構造

各 /system/contract/*type*/*id* ディレクトリに含まれるファイルは、次のとおりです。

ctl
: このファイルをオープンすると、契約 *id* の制御ファイルに対するファイル記述子が返されます。オープン元のプロセスが契約 *id* を保持しておらず、しかもそのオープン元のプロセスが所属するプロセス契約からその契約が継承されていない場合には、このオープンは失敗します。process(4) を参照してください。

  呼び出し元が所有する契約の ctl ファイル記述子では、次の libcontract(3LIB) 呼び出しを行えます。

  ct_ctl_abandon(3contract)
  ct_ctl_newct(3contract)
  ct_ctl_ack(3contract)
  ct_ctl_qack(3contract)

  所有者を持たない契約の ctl ファイル記述子では、次の libcontract(3LIB) 呼び出しを行えます。

  ct_ctl_adopt(3contract)

status
: このファイルをオープンすると、契約 *id* の状態ファイルに対するファイル記述子が返されます。状態ファイル記述子で、次の libcontract(3LIB) 呼び出しを行えます。

ct_status_read(3contract)

「ステータス」を参照してください。

events

このファイルをオープンすると、契約 *id* からのイベントを受信するイベントエンドポイントに対するファイル記述子が返されます。「イベント」を参照してください。

契約のイベントエンドポイントをオープンできるのは、その契約を所有するプロセスと同一の実効ユーザー ID を持つプロセス、その契約の作成者と同一の実効ユーザー ID を持つプロセス、および実効セットに {PRIV_CONTRACT_OBSERVER} が含まれているプロセスだけです。

条項

次の各条項はすべての契約で定義されます。

Cookie
契約作成者が契約の識別子として使用可能な 64 ビット量を指定します。この条項を設定するには、ct_tmpl_set_cookie(3CONTRACT) を使用します。

通知イベントセット
通知イベントとして配信されるイベントを選択します。この条項を設定するには、ct_tmpl_set_informative(3CONTRACT) を使用します。

クリティカルイベントセット
クリティカルイベントとして配信されるイベントを選択します。この条項を設定するには、ct_tmpl_set_critical(3CONTRACT) を使用します。

ステータス

ct_status_read(3CONTRACT) から返されるステータスオブジェクトには、次の各情報が含まれます。

契約 ID
契約の数値 ID。この情報を取得するには、ct_status_get_id(3CONTRACT) を使用します。

契約タイプ
契約のタイプ。文字列として表されます。ct_status_get_type(3CONTRACT) を使って取得します。契約タイプは、/system/contract 配下の対応するサブディレクトリの名前と同じになります。

作成者のゾーン ID
契約を作成したプロセスのゾーン ID。ct_status_get_zoneid(3CONTRACT) を使って取得します。

所有権状態
契約の状態。CTS_OWNED、CTS_INHERITED、CTS_ORPHAN、CTS_DEAD のいずれかで表されます。この情報を取得するには、ct_status_get_state(3CONTRACT) を使用します。

契約保持者
契約の状態が CTS_OWNED である場合、契約を所有するプロセスの ID。契約の状態が CTS_INHERITED である場合、リージェントとして機能している契約の ID。契約の状態が CTS_ORPHAN、CTS_DEAD のいずれかである場合、これは未定義となります。この情報を取得するには、ct_status_get_holder(3CONTRACT) を使用します。

クリティカルイベント数
契約のイベントキュー上で保留状態になっている未確認クリティカルイベントの数。この情報を取得するには、ct_status_get_nevents(3CONTRACT) を使用します。

ネゴシエーション時間
現在の同期ネゴシエーションがタイムアウトするまでの残時間。この情報を取得するには、ct_status_get_ntime(3CONTRACT) を使用します。

ネゴシエーションクォンタム時間
現在のネゴシエーションクォンタムが尽きるまでの残時間。この情報を取得するには、ct_status_get_qtime(3CONTRACT) を使用します。

ネゴシエーションイベント ID
ネゴシエーションタイムアウトを起動したイベントの ID。この情報を取得するには、ct_status_get_nevid(3CONTRACT) を使用します。

Cookie (条項)
契約の Cookie 条項。この情報を取得するには、ct_status_get_cookie(3CONTRACT) を使用します。

通知イベントセット (条項)
契約の通知イベントセット。この情報を取得するには、ct_status_get_informative(3CONTRACT) を使用します。

クリティカルイベントセット (条項)
契約のクリティカルイベントセット。この情報を取得するには、ct_status_get_critical(3CONTRACT) を使用します。

イベント

3 つのイベントエンドポイント /system/contract/*type*/bundle、/system/contract/*type*/pbundle、および /system/contract/*type*/*id*/events には、どれも同じ方法でアクセスできます。

イベントエンドポイントファイル記述子では、次の libcontract(3LIB) インタフェースが使用されます。

ct_event_read(3contract)
ct_event_read_critical(3contract)
ct_event_reset(3contract)
ct_event_next(3contract)

プロセスが複数のイベントエンドポイントを監視しやすくする目的で、イベントエンドポイントに対して poll(2) が呼び出せるようになっています。あるエンドポイントファイル記述子上で受信が可能となっている場合、その記述子に対して POLLIN が設定されます。

ct_event_read(3CONTRACT) から返されたイベントオブジェクトには、次の情報が含まれます。

契約 ID
: イベントを生成した契約の ID。この情報を取得するには、ct_event_get_ctid(3CONTRACT) を使用します。

イベント ID
: 契約イベントの ID。ct_event_get_evid(3CONTRACT) を使用します。

フラグ
: ビットベクトル。通常は CT_ACK と CTE_INFO が含まれます。この情報を取得するには、ct_event_get_flags(3CONTRACT) を使用します。

イベントタイプ
: イベントのタイプ。契約タイプのマニュアルページ内に指定された定数のいずれか、または CT_EV_NEGEND に等しくなります。この情報を取得するには、ct_event_get_type(3CONTRACT) を使用します。

イベントタイプ

次のイベントタイプが定義されています。

CT_EV_NEGEND
: 終了ネゴシエーションの起動後、少し経ってから CT_EV_NEGEND イベントが送信されます。これはネゴシエーションが終了したことを示します。この原因は、処理が取り消されたからか、または処理が成功した可能性があります。成功した場合で、かつ所有者が新しい契約が書き込まれることを要求した場合、これにはその契約の ID が含まれます。

  CT_EV_NEGEND は、契約の通知イベントセットやクリティカルイベントセットに含めることができません。これは常に配信され、常にクリティカルとなります。CT_EV_NEGEND が処理の成功を示している場合、イベントはそれ以上送信されません。契約の所有者は、ct_ctl_abandon(3CONTRACT) を使ってその契約を破棄してください。

  CT_EV_NEGEND イベントには次の情報が含まれます。

  ネゴシエーション ID
  : 終了したネゴシエーションの ID。この情報を取得するには、ct_event_get_nevid(3CONTRACT) を使用します。

  新しい契約 ID
  : 新しく作成された契約の ID。この値は、契約が作成されなかった場合は 0 になります。また、処理が完了しなかった場合は既存の契約の ID になります。この情報を取得するには、ct_event_get_newct(3CONTRACT) を使用します。

ファイル

/system/contract
: すべての契約タイプの一覧

/system/contract/all
: すべての契約 ID のディレクトリ

/system/contract/all/*id*
: 契約 *id* のタイプ固有ディレクトリへのシンボリックリンク

`/system/contract/`*type*
固有タイプのディレクトリ

`/system/contract/`*type*`/templete`
この契約タイプのテンプレート

`/system/contract/`*type*`/bundle`
そのタイプのすべての契約に対する待機ポイント

`/system/contract/`*type*`/pbundle`
オープン元プロセスのそのタイプのすべての契約に対する待機ポイント

`/system/contract/`*type* `/latest`
オープン元の LWP によって作成された最新の *type* 契約の状態

`/system/contract/`*type*`/`*ID*
契約 *ID* のディレクトリ

`/system/contract/`*type*`/`*ID*`/events`
契約 *ID* のイベントに対する待機ポイント

`/system/contract/`*type*`/`*ID*`/ctl`
契約 *ID* の制御ファイル

`/system/contract/`*type*`/`*ID*`/status`
契約 *ID* の状態情報

関連項目

`ctrun`(1), `ctstat`(1), `ctwatch`(1), `chroot`(1M), `close`(2), `ioctl`(2), `open`(2), `poll`(2), `ct_ctl_abandon`(3CONTRACT), `ct_event_get_ctid`(3CONTRACT), `ct_event_get_evid`(3CONTRACT), `ct_event_get_flags`(3CONTRACT), `ct_event_get_nevid`(3CONTRACT), `ct_event_get_newct`(3CONTRACT), `ct_event_get_type`(3CONTRACT), `ct_status_read`(3CONTRACT)`ct_status_get_cookie`(3CONTRACT), `ct_status_get_critical`(3CONTRACT), `ct_status_get_holder`(3CONTRACT), `ct_status_get_id`(3CONTRACT), `ct_status_get_informative`(3CONTRACT), `ct_status_get_nevid`(3CONTRACT), `ct_status_get_nevents`(3CONTRACT), `ct_status_get_ntime`(3CONTRACT), `ct_status_get_qtime`(3CONTRACT), `ct_status_get_state`(3CONTRACT), `ct_status_get_type`(3CONTRACT), `ct_tmpl_set_cookie`(3CONTRACT), `ct_tmpl_set_critical`(3CONTRACT), `ct_tmpl_set_informative`(3CONTRACT), `libcontract`(3LIB), `process`(4), `largefile`(5), `lfcompile`(5), `privileges`(5)

名前

md.tab, md.cf – Solaris ボリュームマネージャのユーティリティファイル

形式

```
/etc/lvm/md.tab
/etc/lvm/md.cf
```

機能説明

/etc/lvm/md.tab は、バッチに類似したモードでメタデバイス、ホットスペア集合、メタデバイス状態データベースの複製を構成するために metainit(1M) および metadb(1M) によって使用されるファイルです。Solaris ボリュームマネージャは、/etc/lvm/md.tab ファイルには構成情報を格納しません。 以下のコマンドを使用すると、このファイルを作成できます。

```
metastat -p > /etc/lvm/md.tab
```

md.tab.4 ファイルの指示に従って、そのファイルを手入力で編集します。同様に、ホットスペアが使用されていない場合は、cp md.cf md.tab コマンドにより、md.tab ファイルとして使用可能なファイルを作成できます。編集は既出の注意事項に従ってください。

md.tab ファイルを使用するときには、ファイル内に指定するそれぞれのメタデバイス、ホットスペア集合、または状態データベースの複製は、一意なエントリである必要があります。エントリには以下のものを含むことができます。単純メタデバイス (ストライプ、連結、連結ストライプ) ミラー、ソフトパーティション、RAID5 メタデバイス、ホットスペア集合、状態データベースの複製。md.tab にはユーザーが入力したエントリだけが含まれるので、システム上である時点のメタデバイス、ホットスペア集合、複製の現在の構成についてはこのファイルを信用しないでください。

タブ、スペース、コメント (# 記号で指定) と行継続子 (バックスラッシュと改行文字を組み合わせたもの) を使用できます。

一般に、メタデバイスは metainit コマンド行で指定した情報に従って設定します。同様に、状態データベースの複製は metadb コマンドで設定します。

コマンド行を使用する代わりに、md.tab ファイルを使用することもできます。メタデバイスと状態データベースの複製は、任意の順序で md.tab ファイルに指定でき、metainit コマンドと metadb コマンドを使用してバッチに類似したモードで起動されます。

md.tab ファイルを編集する場合には、各行で 1 つの完結した構成エントリを指定します。メタデバイスは metainit コマンドで要求されるのと同じ構文を使用して定義します。次に、-a オプションを指定して md.tab ファイルにあるすべてのメタデバイスを有効にするか、または特定の構成エントリに対応するメタデバイス名を指定して metainit コマンドを実行します。

metainit は、-a および -n 両方のフラグを使用して metainit が実行された場合は、既に作成されていると仮定されるボリュームの状態を維持しません。md.tab ファイルの最初の行でデバイス d0 を作成して、後ろの行で d0 の存在を仮定している場合、metainit -a を実行すると、成功する場合がありますが、metainit -an を実行すると、後ろの行は失敗します。

状態データベースの複製は以下のように /etc/lvm/md.tab ファイルに定義されます。mddb *number options* [*slice...*]。mddb *number* は、mddb という文字列の後ろに、状態データベース複製を識別する 2 桁以上の番号を追加したものです。*slice* には、物理スライスを指定します。以下に例を示します。mddb05 /dev/dsk/c0t1d0s2。 /etc/lvm/md.cf は障害からの回復用に使用される構成のバックアップです。ボリュームマネージャ構成が変更されると、いつでもこのファイルは自動的に更新されます (ホットスペアが発生するときを除きます)。このファイルを直接編集してはいけません。

使用例

例 1 連結

以下の例のドライブは、すべて同じサイズ (525 メガバイト) とします。

この例では、4 台のディスクが連結されているメタデバイス /dev/md/dsk/d7 を示します。

```
#
# (4 台のディスクの連結)
#
d7 4 1 c0t1d0s0 1 c0t2d0s0 1 c0t3d0s0 1 c0t4d0s0
```

数値 4 は、この連結に 4 つの独立したストライプがあることを 示しています。各ストライプは 1 つのスライスで構成されているので、各スライスの最初に数値 1 が示されています。上記のすべてのデバイスの最初のディスクセクターには、ディスクラベルが含まれています。デバイス /dev/dsk/c0t2d0s0、/dev/dsk/c0t3d0s0、/dev/dsk/c0t4d0s0 のラベルを保存するために、連結の境界を越えてアクセスを割り当てるときには、メタディスクドライバがこれらのディスクの少なくとも最初のセクターをスキップする必要があります。最初のセクターだけをスキップすると、ディスクのジオメトリが不規則になるので、ディスクの最初のシリンダ全体がスキップされます。このため、上位のファイルシステムソフトウェアによってブロックの割り当てが最適化できます。

例 2 ストライプ

この例は、2 つのストライプで構成されるメタデバイス /dev/md/dsk/d15 を示しています。

```
#
# (2 つのディスクで構成されるストライプ)
#
d15 1 2 c0t1d0s2 c0t2d0s2 -i 32k
```

数値 1 は、1 つのストライプが作成されることを示しています。このストライプは 2 つのスライスで構成されるため、その後に数値 2 が続きます。オプションの -i の後ろには 32k が指定され、飛び越しサイズが 32 キロバイトになることが示されます。飛び越しサイズが指定されなかった場合には、ストライプはデフォルト値の 16 キロバイトを使用します。

例 3 連結ストライプ

この例では、3 台のディスクの 2 つのストライプの連結で構成されるメタデバイス /dev/md/dsk/d75 を示しています。

```
#
# (それぞれ 3 台のディスクで構成される 2 つのストライプの連結)
#
d75 2 3 c0t1d0s2 c0t2d0s2 c0t3d0s2 -i 16k \
      3 c1t1d0s2 c1t2d0s2 c1t3d0s2 -i 32k
```

最初の行では、-i とその後ろの 16k でストライプの飛び越しサイズが 16 キロバイトであることを指定しています。2 番目の組み合わせは、ストライプの飛び越しサイズが 32 キロバイトになることを指定しています。2 番目の組み合わせが 32 キロバイトを指定していなければ、この組み合わせはデフォルトの飛び越し値 16 キロバイトを使用します。3 台のディスクの各組み合わせのブロックは、3 台のディスクにまたがって飛び越しされます。

例 4 ミラー化

この例は、3 つのサブミラーで構成されている 3 面ミラー /dev/md/dsk/d50 を示します。このミラーには、既存のデータは含まれていません。

```
#
# (ミラー)
#
d50 -m d51
d51 1 1 c0t1d0s2
d52 1 1 c0t2d0s2
d53 1 1 c0t3d0s2
```

この例では、最初に -m オプションを使用して 1 面ミラーが定義されます。この 1 面ミラーは、サブミラー d51 で構成されます。ほかの 2 つのサブミラー d52 と d53 は、metattach コマンドを使用して後で追加されます。この例の読み取りおよび書き込みのデフォルトオプションでは、すべてのサブミラーに対して、読み取りはラウンドロビン (巡回的) に処理され、書き込みは並列処理されます。したがって、/etc/lvm/md.tab ファイル内でミラーを定義する順序は、重要でありません。

例 5 RAID5

この例は、3 つのスライスで構成される RAID5 メタデバイス d80 を示します。

```
#
# (RAID デバイス)
#
d80 -r c0t1d0s1 c1t0d0s1 c2t0d0s1 -i 20k
```

例 5 RAID5　　*(続き)*

この例で、RAID5 メタデバイスは -r オプションによって飛び越しサイズ 20 キロバイトで定義されます。データとパリティーセグメントは、`c0t1d0s1`、`c1t0d0s1`、`c2t0d0s1` のスライスにまたがってストライプされます。

例 6 ソフトパーティション

この例は、9G バイトのディスク全体を再フォーマットするソフトパーティション `d85` を示しています。状態データベースの複製用に予約されている領域である、スライス 7 が保持する数メガバイトを除き、スライス 0 はディスクをすべて占有します。スライス 7 は最小 4M バイトですが、ディスクのジオメトリに応じて大きくすることもできます。`d85` は `c3t4d0s0` 上にあります。

ドライブがディスクセットに追加される時にドライブのパーティションが再設定されるのは、スライス 7 が正しく設定されていない場合のみです。各ドライブのわずかな一部が、ボリュームマネージャで使用するためスライス 7 で予約されます。各ドライブの残り容量は、スライス 0 に配置されます。パーティションを再設定した後、ディスク上の既存のデータは失われます。ディスクセットにドライブを追加した後、必要に応じてドライブをパーティションを再設定することができます。しかし、スライス 7 は、移動または削除したり、ほかのパーティションと重複させないでください。

ソフトパーティションのオフセットと範囲を手入力で指定することはお勧めできません。この例は、ファイルが自動的に生成された場合にファイルをよりよく理解できるよう、また完全を期すために含まれています。

```
#
# (ソフトパーティション)
d85 -p -e c3t4d0 9g
```

この例では、ソフトパーティションの作成と、状態データベースの複製に必要な容量で、ディスク `c3t4d0` の 9G バイトすべてを占有しています。

例 7 ソフトパーティション

この例では、2 つの範囲を持つソフトパーティションの再作成に使用されるコマンドを示します。2 つの範囲は、オフセット 20483 から始まり 20480 ブロックの幅がある第 1 の範囲と、135398 から始まり 20480 ブロックの幅がある第 2 の範囲です。

```
#
# (ソフトパーティション)
#
d1 -p c0t3d0s0 -o 20483 -b 20480 -o 135398 -b 20480
```

例 8 ホットスペア

この例は、3 つのサブミラーと 3 つのホットスペア集合で構成される 3 面ミラー `/dev/md/dsk/d10` を示します。

例 8 ホットスペア　　(続き)

```
#
# (ミラーとホットスペア)
#
d10 -m d20
d20 1 1 c1t0d0s2 -h hsp001
d30 1 1 c2t0d0s2 -h hsp002
d40 1 1 c3t0d0s2 -h hsp003
hsp001 c2t2d0s2 c3t2d0s2 c1t2d0s2
hsp002 c3t2d0s2 c1t2d0s2 c2t2d0s2
hsp003 c1t2d0s2 c2t2d0s2 c3t2d0s2
```

この例では、最初に -m オプションで 1 面ミラーが定義されます。サブミラーは、metattach(1M) コマンドによって、後で追加します。使用するホットスペア集合は、-h オプションによってサブミラーに結合します。この例では、3 台のディスクがホットスペア集合に定義され、別々のホットスペアとして使用されています。ホットスペア集合には、hsp001、hsp002、hsp003 という名前が付けられています。コンポーネントごとにホットスペアを 1 つ割り当てるのではなく、3 つのホットスペア集合を設定すると、ハードウェアの可用度を最大にすることができます。このように構成すると、最初に最も適したホットスペアが選択されるように指定することができ、利用可能なホットスペアを増やすことによってディスクの使用効率が高くなります。エントリの末尾には、使用するホットスペアを定義します。ホットスペアとメタデバイスを対応付けるために md.tab ファイルを使用するときには、ホットスペア集合を対応の前に存在させる必要はありません。md.tab ファイルを使用する場合、ボリュームマネージャがメタデバイスやホットスペアを作成する順序を考慮します。

例 9 状態データベース複製

この例では、3 台のディスクを持つサーバー上で、初期状態データベースと 3 つの複製を設定します。

```
#
# (状態データベースと複製)
#
mddb01 -c 3 c0t1d0s0 c0t2d0s0 c0t3d0s0
```

この例では、3 つの状態データベースの複製が 3 つのスライスにそれぞれ保存されます。上記のエントリを /etc/lvm/md.tab ファイルに入力した場合は、-a オプションと -f オプションの両方を指定して metadb コマンドを実行してください。たとえば、以下のコマンドを入力すると、3 つのスライス上に 1 つの状態データベースの複製が作成されます。

```
# metadb -a -f mddb01
```

ファイル

- /etc/lvm/md.tab
- /etc/lvm/md.cf

関連項目

mdmonitord(1M), metaclear(1M), metadb(1M), metadetach(1M), metahs(1M), metainit(1M), metaoffline(1M), metaonline(1M), metaparam(1M), metarecover(1M), metarename(1M), metareplace(1M), metaroot(1M), metassist(1M), metaset(1M), metastat(1M), metasync(1M), metattach(1M), md.cf(4), mddb.cf(4), attributes(5), md(7D)

Solaris ボリュームマネージャの管理

制限事項

ミラー化を再帰的に行うことはできません。つまり、ミラーを別のミラーの定義で使用することはできません。

また、ロギングを再帰的に行うこともできません。

ストライプと RAID5 メタデバイスに含まれるものは、スライスまたはソフトパーティションだけでなければなりません。

RAID5 メタデバイスをミラー化することはできません。

ソフトパーティションはスライス上に直接構築したり (アプリケーションにより直接アクセス可能な) 最上位レベルに配置したりできますが、上下にその他のメタデバイスが存在する位置に配置することはできません。

注意事項

トランスメタデバイスは UFS ロギングに置き換えられています。既存のトランスデバイスはロギングを行わず、基礎デバイスにデータを直接渡します。UFS ロギングの詳細については、mount_ufs(1M) を参照してください。

名前

mddb.cf – メタデバイス状態データベースの複製の位置

形式

`/etc/lvm/mddb.cf`

機能説明

`/etc/lvm/mddb.cf` ファイルは、`metadb`(1M) コマンドの起動時に作成されます。このファイルは絶対に直接編集しないでください。

`/etc/lvm/mddb.cf` ファイルは、`metainit`(1M) コマンドの実行時に、メタデバイス状態データベースの複製の位置を検出するために使用されます。`metadb` コマンドは実行されるたびにこのファイルを作成および更新します。同じ情報は `/kernel/drv/md.conf` ファイルにも入力されます。

`/etc/lvm/mddb.cf` ファイルには、各メタデバイス状態データベースの複製ごとに、一意のエントリがあります。各エントリには、複製の保存先であるブロック物理デバイスに関連付けられた、ドライバユニット番号とマイナーユニット番号が含まれています。また、各エントリには、マスターブロックのブロック番号も含まれています。マスターブロックには、複製内のほかのすべてのブロックのリストが含まれています。

`/etc/lvm/mddb.cf` ファイル内のエントリの形式は、以下のとおりです。 *driver_name minor_t daddr_t checksum*。ここで *driver_name* と *minor_t* は、この複製を保存する物理デバイスのデバイス番号です。*daddr_t* は、ディスクのブロックアドレスです。*checksum* は、エントリが破壊されていないかどうかを確認するために使用します。ポンド記号 (#) の後には、コメントが続きます。

使用例

例 1 ファイルの例

以下に、`mddb.cf` ファイルの例を示します。

```
#metadevice database location file do not hand edit
#driver minor_t daddr_t device id       checksum
sd      152     16      id1,sd@SSEAGATE_JDD288110MC9LH/a        -2613
```

上記例の `daddr_t` の値は、指定されたパーティションの開始点からのオフセットが、そのパーティションの開始点から 16 ディスクブロックであることを示しています。

ファイル

`/etc/lvm/mddb.cf`

`/kernel/drv/md.conf`

関連項目

`mdmonitord`(1M), `metaclear`(1M), `metadb`(1M), `metadetach`(1M), `metahs`(1M), `metainit`(1M), `metaoffline`(1M), `metaonline`(1M), `metaparam`(1M), `metarecover`(1M), `metarename`(1M), `metareplace`(1M), `metaroot`(1M), `metassist`(1M), `metaset`(1M), `metastat`(1M), `metasync`(1M), `metattach`(1M), `md.cf`(4), `md.tab`(4), `attributes`(5), `md`(7D)

Solaris ボリュームマネージャの管理

名前

md.tab, md.cf – Solaris ボリュームマネージャのユーティリティファイル

形式

```
/etc/lvm/md.tab
/etc/lvm/md.cf
```

機能説明

`/etc/lvm/md.tab` は、バッチに類似したモードでメタデバイス、ホットスペア集合、メタデバイス状態データベースの複製を構成するために `metainit`(1M) および `metadb`(1M) によって使用されるファイルです。Solaris ボリュームマネージャは、`/etc/lvm/md.tab` ファイルには構成情報を格納しません。 以下のコマンドを使用すると、このファイルを作成できます。

```
metastat -p > /etc/lvm/md.tab
```

`md.tab.4` ファイルの指示に従って、そのファイルを手入力で編集します。同様に、ホットスペアが使用されていない場合は、`cp md.cf md.tab` コマンドにより、`md.tab` ファイルとして使用可能なファイルを作成できます。編集は既出の注意事項に従ってください。

`md.tab` ファイルを使用するときには、ファイル内に指定するそれぞれのメタデバイス、ホットスペア集合、または状態データベースの複製は、一意なエントリである必要があります。エントリには以下のものを含むことができます。単純メタデバイス (ストライプ、連結、連結ストライプ) ミラー、ソフトパーティション、RAID5 メタデバイス、ホットスペア集合、状態データベースの複製。`md.tab` にはユーザーが入力したエントリだけが含まれるので、システム上である時点のメタデバイス、ホットスペア集合、複製の現在の構成についてはこのファイルを信用しないでください。

タブ、スペース、コメント (`#` 記号で指定) と行継続子 (バックスラッシュと改行文字を組み合わせたもの) を使用できます。

一般に、メタデバイスは `metainit` コマンド行で指定した情報に従って設定します。同様に、状態データベースの複製は `metadb` コマンドで設定します。

コマンド行を使用する代わりに、`md.tab` ファイルを使用することもできます。メタデバイスと状態データベースの複製は、任意の順序で `md.tab` ファイルに指定でき、`metainit` コマンドと `metadb` コマンドを使用してバッチに類似したモードで起動されます。

`md.tab` ファイルを編集する場合には、各行で 1 つの完結した構成エントリを指定します。メタデバイスは `metainit` コマンドで要求されるのと同じ構文を使用して定義します。次に、`-a` オプションを指定して `md.tab` ファイルにあるすべてのメタデバイスを有効にするか、または特定の構成エントリに対応するメタデバイス名を指定して `metainit` コマンドを実行します。

`metainit` は、`-a` および `-n` 両方のフラグを使用して `metainit` が実行された場合は、既に作成されていると仮定されるボリュームの状態を維持しません。`md.tab` ファイルの最初の行でデバイス `d0` を作成して、後ろの行で d0 の存在を仮定している場合、`metainit -a` を実行すると、成功する場合がありますが、`metainit -an` を実行すると、後ろの行は失敗します。

状態データベースの複製は以下のように /etc/lvm/md.tab ファイルに定義されます。mddb *number options* [*slice...*]。mddb *number* は、mddb という文字列の後ろに、状態データベース複製を識別する 2 桁以上の番号を追加したものです。*slice* には、物理スライスを指定します。以下に例を示します。mddb05 /dev/dsk/c0t1d0s2。 /etc/lvm/md.cf は障害からの回復用に使用される構成のバックアップです。ボリュームマネージャ構成が変更されると、いつでもこのファイルは自動的に更新されます (ホットスペアが発生するときを除きます)。このファイルを直接編集してはいけません。

使用例

例 1 連結

以下の例のドライブは、すべて同じサイズ (525 メガバイト) とします。

この例では、4 台のディスクが連結されているメタデバイス /dev/md/dsk/d7 を示します。

```
#
# (4 台のディスクの連結)
#
d7 4 1 c0t1d0s0 1 c0t2d0s0 1 c0t3d0s0 1 c0t4d0s0
```

数値 4 は、この連結に 4 つの独立したストライプがあることを 示しています。各ストライプは 1 つのスライスで構成されているので、各スライスの最初に数値 1 が示されています。上記のすべてのデバイスの最初のディスクセクターには、ディスクラベルが含まれています。デバイス /dev/dsk/c0t2d0s0、/dev/dsk/c0t3d0s0、/dev/dsk/c0t4d0s0 のラベルを保存するために、連結の境界を越えてアクセスを割り当てるときには、メタディスクドライバがこれらのディスクの少なくとも最初のセクターをスキップする必要があります。最初のセクターだけをスキップすると、ディスクのジオメトリが不規則になるので、ディスクの最初のシリンダ全体がスキップされます。このため、上位のファイルシステムソフトウェアによってブロックの割り当てが最適化できます。

例 2 ストライプ

この例は、2 つのストライプで構成されるメタデバイス /dev/md/dsk/d15 を示しています。

```
#
# (2 つのディスクで構成されるストライプ)
#
d15 1 2 c0t1d0s2 c0t2d0s2 -i 32k
```

数値 1 は、1 つのストライプが作成されることを示しています。このストライプは 2 つのスライスで構成されるため、その後に数値 2 が続きます。オプションの -i の後ろには 32k が指定され、飛び越しサイズが 32 キロバイトになることが示されます。飛び越しサイズが指定されなかった場合には、ストライプはデフォルト値の 16 キロバイトを使用します。

例 3 連結ストライプ

この例では、3 台のディスクの 2 つのストライプの連結で構成されるメタデバイス /dev/md/dsk/d75 を示しています。

```
#
# (それぞれ 3 台のディスクで構成される 2 つのストライプの連結)
#
d75 2 3 c0t1d0s2 c0t2d0s2 c0t3d0s2 -i 16k \
      3 c1t1d0s2 c1t2d0s2 c1t3d0s2 -i 32k
```

最初の行では、-i とその後ろの 16k でストライプの飛び越しサイズが 16 キロバイトであることを指定しています。2 番目の組み合わせは、ストライプの飛び越しサイズが 32 キロバイトになることを指定しています。2 番目の組み合わせが 32 キロバイトを指定していなければ、この組み合わせはデフォルトの飛び越し値 16 キロバイトを使用します。3 台のディスクの各組み合わせのブロックは、3 台のディスクにまたがって飛び越しされます。

例 4 ミラー化

この例は、3 つのサブミラーで構成されている 3 面ミラー /dev/md/dsk/d50 を示します。このミラーには、既存のデータは含まれていません。

```
#
# (ミラー)
#
d50 -m d51
d51 1 1 c0t1d0s2
d52 1 1 c0t2d0s2
d53 1 1 c0t3d0s2
```

この例では、最初に -m オプションを使用して 1 面ミラーが定義されます。この 1 面ミラーは、サブミラー d51 で構成されます。ほかの 2 つのサブミラー d52 と d53 は、metattach コマンドを使用して後で追加されます。この例の読み取りおよび書き込みのデフォルトオプションでは、すべてのサブミラーに対して、読み取りはラウンドロビン (巡回的) に処理され、書き込みは並列処理されます。したがって、/etc/lvm/md.tab ファイル内でミラーを定義する順序は、重要でありません。

例 5 RAID5

この例は、3 つのスライスで構成される RAID5 メタデバイス d80 を示します。

```
#
# (RAID デバイス)
#
d80 -r c0t1d0s1 c1t0d0s1 c2t0d0s1 -i 20k
```

例 5 RAID5　*(続き)*

この例で、RAID5 メタデバイスは -r オプションによって飛び越しサイズ 20 キロバイトで定義されます。データとパリティーセグメントは、c0t1d0s1、c1t0d0s1、c2t0d0s1 のスライスにまたがってストライプされます。

例 6 ソフトパーティション

この例は、9G バイトのディスク全体を再フォーマットするソフトパーティション d85 を示しています。状態データベースの複製用に予約されている領域である、スライス 7 が保持する数メガバイトを除き、スライス 0 はディスクをすべて占有します。スライス 7 は最小 4M バイトですが、ディスクのジオメトリに応じて大きくすることもできます。d85 は c3t4d0s0 上にあります。

ドライブがディスクセットに追加される時にドライブのパーティションが再設定されるのは、スライス 7 が正しく設定されていない場合のみです。各ドライブのわずかな一部が、ボリュームマネージャで使用するためスライス 7 で予約されます。各ドライブの残り容量は、スライス 0 に配置されます。パーティションを再設定した後、ディスク上の既存のデータは失われます。ディスクセットにドライブを追加した後、必要に応じてドライブをパーティションを再設定することができます。しかし、スライス 7 は、移動または削除したり、ほかのパーティションと重複させないでください。

ソフトパーティションのオフセットと範囲を手入力で指定することはお勧めできません。この例は、ファイルが自動的に生成された場合にファイルをよりよく理解できるよう、また完全を期すために含まれています。

```
#
# (ソフトパーティション)
d85 -p -e c3t4d0 9g
```

この例では、ソフトパーティションの作成と、状態データベースの複製に必要な容量で、ディスク c3t4d0 の 9G バイトすべてを占有しています。

例 7 ソフトパーティション

この例では、2 つの範囲を持つソフトパーティションの再作成に使用されるコマンドを示します。2 つの範囲は、オフセット 20483 から始まり 20480 ブロックの幅がある第 1 の範囲と、135398 から始まり 20480 ブロックの幅がある第 2 の範囲です。

```
#
# (ソフトパーティション)
#
d1 -p c0t3d0s0 -o 20483 -b 20480 -o 135398 -b 20480
```

例 8 ホットスペア

この例は、3 つのサブミラーと 3 つのホットスペア集合で構成される 3 面ミラー /dev/md/dsk/d10 を示します。

```
#
# (ミラーとホットスペア)
#
d10 -m d20
d20 1 1 c1t0d0s2 -h hsp001
d30 1 1 c2t0d0s2 -h hsp002
d40 1 1 c3t0d0s2 -h hsp003
hsp001 c2t2d0s2 c3t2d0s2 c1t2d0s2
hsp002 c3t2d0s2 c1t2d0s2 c2t2d0s2
hsp003 c1t2d0s2 c2t2d0s2 c3t2d0s2
```

この例では、最初に -m オプションで 1 面ミラーが定義されます。サブミラーは、metattach(1M) コマンドによって、後で追加します。使用するホットスペア集合は、-h オプションによってサブミラーに結合します。この例では、3 台のディスクがホットスペア集合に定義され、別々のホットスペアとして使用されています。ホットスペア集合には、hsp001、hsp002、hsp003 という名前が付けられています。コンポーネントごとにホットスペアを 1 つ割り当てるのではなく、3 つのホットスペア集合を設定すると、ハードウェアの可用度を最大にすることができます。このように構成すると、最初に最も適したホットスペアが選択されるように指定することができ、利用可能なホットスペアを増やすことによってディスクの使用効率が高くなります。エントリの末尾には、使用するホットスペアを定義します。ホットスペアとメタデバイスを対応付けるために md.tab ファイルを使用するときには、ホットスペア集合を対応の前に存在させる必要はありません。md.tab ファイルを使用する場合、ボリュームマネージャがメタデバイスやホットスペアを作成する順序を考慮します。

例 9 状態データベース複製

この例では、3 台のディスクを持つサーバー上で、初期状態データベースと 3 つの複製を設定します。

```
#
# (状態データベースと複製)
#
mddb01 -c 3 c0t1d0s0 c0t2d0s0 c0t3d0s0
```

この例では、3 つの状態データベースの複製が 3 つのスライスにそれぞれ保存されます。上記のエントリを /etc/lvm/md.tab ファイルに入力した場合は、-a オプションと -f オプションの両方を指定して metadb コマンドを実行してください。たとえば、以下のコマンドを入力すると、3 つのスライス上に 1 つの状態データベースの複製が作成されます。

```
# metadb -a -f mddb01
```

ファイル

- /etc/lvm/md.tab
- /etc/lvm/md.cf

関連項目

mdmonitord(1M), metaclear(1M), metadb(1M), metadetach(1M), metahs(1M), metainit(1M), metaoffline(1M), metaonline(1M), metaparam(1M), metarecover(1M), metarename(1M), metareplace(1M), metaroot(1M), metassist(1M), metaset(1M), metastat(1M), metasync(1M), metattach(1M), md.cf(4), mddb.cf(4), attributes(5), md(7D)

Solaris ボリュームマネージャの管理

制限事項

ミラー化を再帰的に行うことはできません。つまり、ミラーを別のミラーの定義で使用することはできません。

また、ロギングを再帰的に行うこともできません。

ストライプと RAID5 メタデバイスに含まれるものは、スライスまたはソフトパーティションだけでなければなりません。

RAID5 メタデバイスをミラー化することはできません。

ソフトパーティションはスライス上に直接構築したり (アプリケーションにより直接アクセス可能な) 最上位レベルに配置したりできますが、上下にその他のメタデバイスが存在する位置に配置することはできません。

注意事項

トランスメタデバイスは UFS ロギングに置き換えられています。既存のトランスデバイスはロギングを行わず、基礎デバイスにデータを直接渡します。UFS ロギングの詳細については、mount_ufs(1M) を参照してください。

名前

power.conf – 電源管理システムの設定情報ファイル

形式

/etc/power.conf

機能説明

power.conf ファイルは、電源管理の設定値を初期化するために、電源管理設定プログラム (pmconfig(1M)) によって使用されます。このファイルを修正する場合は、手動で pmconfig(1M) を実行して変更を適用してください。

dtpower(1M) GUI を使用すると、このファイルで使用可能なパラメータのサブセットを設定することができます。簡単にパラメータを設定するには、dtpower(1M) を使用することをお勧めします。電源管理を無効にする方法については、dtpower(1M) の適切なセクションを参照してください。

電源管理システムには、個々のデバイスの管理とシステム全体の管理の 2 つの管理があります。デバイスが複数の電源レベルをサポートし、デバイスがアイドル状態になった時にデバイスの電源を節約するためにデバイスドライバがカーネルにより提供された電源管理インタフェースを使用する場合、個々のデバイスは電源管理されます。

power.conf ファイルのすべてのエントリは、ファイルに表示されている順番で処理されます。

自動デバイス電源管理

autopm エントリが有効になっている場合、自動デバイス電源管理インタフェースを使用するドライバを持つデバイスは自動的に電源管理されます。autopm エントリは、このセクションの終わり付近に記述されています。pm-components プロパティは、デバイスドライバの電源管理モデルを電源管理フレームワークに記述します。詳細については、pm-components(9P) を参照してください。

threshold 時間の間、部品がある電源レベルでアイドル状態になっている場合、部品の電源レベルは次に低い電源レベル (ある場合) まで低下します。複数の部品を実装するデバイスでは、部品は個別に電源管理されます。

自動的に電源管理されている部品のデフォルトのしきい値は、電源管理システムのフレームワークによりシステムのアイドル状態 *threshold* に基づいて計算されます。デフォルトで、デバイスのすべての部品は、システムのアイドル状態の*threshold* 値の間アイドル状態になっていると、電源が切断されます。デフォルトのシステムのアイドル状態の *threshold* は、米国環境保護局 (EPA) の Energy Star Memorandum of Understanding で決められています。詳細は、このマニュアルページの「注意事項」セクションを参照してください。

システムのアイドル状態の *threshold* を設定するには、以下のエントリのいずれかを使用します。

system-threshold *threshold*

system-threshold always-on

threshold は、h、m、s を付記して表されるシステムのアイドル状態のしきい値を示します (数値だけが入力された場合はデフォルトで秒数を表します)。always-on が指定されると、デフォルトですべてのデバイスはフルパワーのままになります。

電源管理システムのフレームワークにより与えられたデフォルトのデバイス部品のしきい値を変更するには、device-thresholds エントリを使用します。device-thresholds エントリは、特定の自動電源管理されているデバイスのしきい値を設定、またはそのデバイスの自動電源管理システムを無効にします。

device-thresholds は、以下の形式で指定します。

device-thresholds *phys_path (threshold ...) ...*

device-thresholds *phys_path threshold*

device-thresholds *phys_path* always-on

phys_path は、特定のデバイスの物理パス (libdevinfo(3LIB)) を指定します。たとえば、/pci@8,600000/scsi@4/ssd@w210000203700c3ee,0はディスクの物理パスです。/devices ツリーへのシンボリックリンク (/dev/dsk/c1t1d0s0 など) も可能です。しきい値は特定のデバイスだけに適用されます (あるいはデバイスを常に適用状態にします)。

上記の 1 番目の形式では各 *threshold* 値は、電源がこの部品の次の電源レベルに下げられるまでの現在の電源レベルでアイドル状態が継続する時間を、時、分、秒数で指定します (h、m、s で単位を表しますが、デフォルトでは秒数を表します)。( ) 内には、部品ごとのグループしきい値を示しますが、最初 (一番左) のグループには、部品 0、次のグループには、部品 1 が適用されます。1 つのグループ内の最後 (一番右側) の数字は、最高レベルの 1 つ下のレベルに下げられるまでのもっとも高い電源レベルでアイドル状態が継続する時間を表します。最初 (一番左) の数字は、もっとも低い電源レベルに下げられるまでのもっとも低い電源レベルの 1 つ上の電源レベルでアイドル状態が継続する時間を表します。

グループ数が、デバイス (pm-components(9P) プロパティ) でエクスポートされた部品数と一致しない、またはグループのしきい値の数が対応する部品がサポートする電源レベルの数より少ない場合、エラーメッセージが表示されて、エントリは無視されます。

たとえば、*xfb* と呼ばれるデバイスが、部品 *Frame Buffer* と *Monitor* をエクスポートしたとすると、*Frame Buffer* には、On とOff の 2 つの電源レベルが考えられ、*Monitor* には、Off、Suspend、Standby、On の 4 つレベルが考えられます。

device-thresholds エントリは以下のようになります。

device-thresholds */pci@f0000/xfb@0 (0) (3m 5m 15m)*

上記の形式では特殊な *xfb* カードの *Monitor* 部品の *threshold* 時間として、On から Standby になるのに 15 分、Standby から Suspend になるのに 5 分、Suspend から Off になるのに 3 分を設定します。*Frame Buffer* が On から Off に変わるしきい値は 0 秒です。

2 番目の形式では、() なしで *threshold* 値を 1 つ指定します。*threshold* は、デバイスがアイドル状態の場合デバイス全体の電源が切断されるまでの最大時間を表します。システムはデバイス間の内部依存性については認識していないため、すべてのデバイスがアイドル状態の場合、あるデバイスが実際に指定された *threshold* 値よりも早く電源が切断される場合がありますが、設定された時間よりあとに電源が切断されることはありません。

3 番目の形式では、デバイスのすべての部品がフルパワーのままになります。

デバイス電源管理エントリは、デバイスを直接制御するユーザープロセスがない場合にのみ有効です。たとえば、X ウィンドウシステムは直接フレームバッファーを制御し、`power.conf` ファイルのエントリは、X ウィンドウシステムが稼働していない場合にのみ有効です。

デバイス間の依存性についても定義されることがあります。デバイスがほかのデバイスに依存するとは、ほかのデバイスのすべての部品の電源が切断されて初めて電源レベルが下げられるということです。 依存性は、以下の形式のエントリで示されます。

`device-dependency` *dependent_phys_path phys_path [phys_path ...]*

dependent_phys_path は、ほかのデバイスに依存しているデバイスのパス名 (上記同様) です。*phys_path* エントリには、そのデバイスに依存されているデバイスを指定します。`/devices` ツリーへのシンボリックリンク (`/dev/fb` など) も可能です。このエントリは、デバイスの論理的依存を示すためだけに必要です。論理的依存デバイスとは、電源管理されているデバイスに物理的に接続されていないデバイスです (ディスプレイとキーボードなど)。物理的依存デバイスは自動的に依存しているとみなされるため、エントリにいれる必要はありません。

物理パスを使用して依存性をリストするだけでなく、次の構文でプロパティの依存性を指定して、任意のデバイスグループ間に依存性を作成することもできます。

`device-dependency-property` *property phys_path* [*phys_path* ...]

property で指定したプロパティをエクスポートする各デバイスは、*phys_path* で指定したデバイスに依存します。*phys_path* のパス名の場合と同様に、`/devices` ツリーへのシンボリックリンク (`/dev/fb` など) も可能です。

たとえば、以下の例を考えてみましょう。

```
# This entry keeps removable media from being powered down unless the
# console framebuffer and monitor are powered down
# (See removable-media(9P))
#
device-dependency-property removable-media /dev/fb
```

これによって、`removable-media` という名前の boolean 型のプロパティをエクスポートするすべてのデバイスは、コンソールのフレームバッファが稼動している間は動作することになります (`removable-media`(9P) を参照)。

autopm エントリは、システム全体に対して自動デバイス電源管理システムを有効にしたり、無効にしたりするのに使用されることもあります。autopm エントリは、以下の形式で指定します。

autopm *behavior*

以下に、指定することができる *behavior* 値とその意味を示します。

default	システムの動作はそのモデルに依存します。米国環境保護局の Energy Star Memorandum of Understanding #3 に準拠するデスクトップモデルは、自動的にデバイス電源管理システムが有効になり、ほかのすべてのモデルは有効になりません。詳細は、「注意事項」セクションを参照してください。
enable	このエントリが見つかると、自動デバイス電源管理は再開します。
disable	このエントリが見つかると、自動デバイス電源管理は停止します。

システム電源管理

システム電源管理エントリは、保存停止・復元再開機能を使用してシステム全体の電源管理を制御します。

システムが保存停止になった場合、電源が切断される前に現在の状態はディスクに保存されます。再起動時、システムは自動的に操作を再開して、保存停止前の状態に復旧します。

システムは、保存停止・復元再開機能を使用し以下の形式でエントリを指定して、自動的に停止するように設定することができます。

autoshutdown *idle_time start_time finish_time behavior*

idle_time は、システムが自動的に停止されるまでに必要なシステムのアイドル時間を分単位で指定します。システムのアイドル状態は、システムが非アクティブかどうかで判定されます。また以下に示すように設定できます。

start_time と *finish_time* (両方とも hh:mm 形式) は、システムが自動的に停止されるまでの時間帯を指定します。これらの時刻は、その日の始まり (12:00 a.m.) をもとに計算されます。 *finish_time* が、 *start_time* と同じかそれより早い場合、時間は午前 0 時から *finish_time* までと *start_time* から次の午前 0 時までです。継続した操作を指定するには、*finish_time* は、 *start_time* と同じに設定することもできます。

以下に、指定することができる *behavior* 値とその意味を示します。

shutdown	システムが *idle_time* 値で指定された時間の間アイドル状態にあり、かつ現在の時刻が *start_time* 時刻と *finish_time* 時刻の間にある場合にシステムは自動停止します。
noshutdown	システムが自動停止することはありません。

`autowakeup` ハードウェアが自動立ち上がり機能を持つ場合、システムは `shutdown` が指定された場合と同様に停止し、時刻が *finish_time* 時刻になると自動的に再開します。

`default` システムの動作はそのモデルに依存します。米国環境保護局 (EPA) の Energy Star Memorandum of Understanding #2 ガイドラインに準拠したデスクトップモデルは、自動的に `shutdown` が有効になります (*behavior* フィールドが `shutdown` に設定された場合と同様に動作します) が、ほかのモデルは自動的に停止されません。「注意事項」を参照してください。

`unconfigured` システムは自動的に停止しません。システムのインストールまたはアップグレードの直後は、このフィールドの値は次回の再起動時に変更されます

以下の形式を使用して、システムのアイドル状態を設定することができます。

idleness_parameter value

idleness_parameter に指定できるのは以下のものです。

`ttychars` *idleness_parameter* が `ttychars` の場合、*value* フィールドは、システムがアイドル状態にあるとみなされることを可能にしながら `ldterm` モジュールを通過することができる tty文字の最大数として解釈されます。エントリが設定されていない場合、この値はデフォルトで `0` になります。

`loadaverage` *idleness_parameter* フィールドが `loadaverage` の場合、(浮動小数点の) *value* フィールドは、システムがアイドル状態にあるとみなされることを可能にしながら参照できる最大平均負荷率として解釈されます。エントリが設定されていない場合、この値はデフォルトで `0.04` になります。

`diskreads` *idleness_parameter* フィールドが `diskreads` の場合、*value* フィールドは、システムがアイドル状態にあるとみなされることを可能にしながらシステムが実行することができるディスクの読み取り操作の最大数として解釈されます。エントリが設定されていない場合、この値はデフォルトで `0` になります。

`nfsreqs` *idleness_parameter* フィールドが `nfsreqs` の場合、*value* フィールドは、システムがアイドル状態にあるとみなされることを可能にしながらシステムが送信または受信することができる NFS 要求の最大数として解釈されます。エントリが設定されていない場合、この値はデフォルトで `0` になります。空の要求、アクセス要求および gettattr

要求は、この数から除外されます。エントリが設定されていない場合、この値はデフォルトで `0` になります。

`idlecheck`

idleness_parameter フィールドが `idlecheck` の場合、*value* フィールドが特殊デバイス名 idlecheck を含む場合には、device_name フィールドの次には、システムがアイドル状態であるかどうかを判定するプログラムのパス名が続く必要があります。`autoshutdown` が有効で、かつ、コンソールキーボード、マウス、tty、CPU (平均負荷率によって示される)、ネットワーク (NFS 要求によって測定される)、ディスク (読み取り回数によって測定される) が、上記で指定された `autoshutdown` エントリに指定された時間アイドル状態 にあり、かつ、時刻が `start` 時刻と finish 時刻の間にある場合、このプログラムは他のアイドル状態条件を調べるために実行されます。上記の `autoshutdown` エントリに指定されたアイドル時間の値は、環境変数 `PM_IDLETIME` によってプログラムに渡されます。プロセスは、システムがアイドル状態にあるとこのプロセスがみなした時間量 (分) を表す終了コードで終了します。

デフォルトの *idlecheck* エントリはありません。

システムが保存停止すると、現在のシステムの状態がディスクの状態ファイルに保存されます。状態ファイルの場所を変更するには、以下の形式のエントリを編集します。

`statefile` *pathname*

pathname は、`/dev/dsk/c1t0d0s3` などのブロック型特殊ファイルか、ローカルの `ufs` ファイルを示します。*pathname* がブロック型特殊ファイルを指定している場合、そのブロック型特殊ファイルにファイルシステムがマウントされていないかぎりシンボリックリンクであってもかまいません。*pathname* のローカル `ufs` ファイルを指定値手いる場合、シンボリックリンクにはできません。このファイルがない場合、`suspend` 状態の間にこのファイルが作成されます。パスを構成するすべてのディレクトリは、事前に存在している必要があります。

状態ファイルの実サイズは、システムのメモリーの容量、使用されている読み込み可能なドライバやモジュール、実行されているプロセスの数と種類、「ロックダウン」されたユーザー記憶領域など、さまざまな要素によって左右されます。状態ファイルは、10M バイト以上の空き容量を持つファイルシステムに置くことをお勧めします。起動時に状態ファイルがない場合は、システムにより必要な新しいエントリが自動的に作成されます。

属性

以下の属性については、`attributes`(5) を参照してください。

属性タイプ	属性値
使用条件	SUNWpmr
インタフェースの安定性	Evolving

関連項目

pmconfig(1M), powerd(1M), sys-unconfig(1M), uadmin(2), libdevinfo(3LIB), attributes(5), cpr(7), ldterm(7M), pm(7D), pm-components(9P), removable-media(9P)

『Writing Device Drivers』

『電源管理システム ユーザーマニュアル』

注意事項

1995 年 10 月 1 日以降1999 年 7 月 1 日より前に出荷された SPARC デスクトップモデルは、米国環境保護局 (EPA) の Energy Star Memorandum of Understanding #2 ガイドラインに準拠しています。 またデフォルトで 30 分間アイドル状態が続くと自動で停止します。この機能は、これらのマシンで shutdown と同様に動作するデフォルトのキーワードである autoshutdown により実行されます。システムを設置し、再起動する際や、sys-unconfig(1M) を使用してシステムを構成解除した直後の起動の際に、このデフォルト設定を確認するメッセージが表示されます。

1999 年 7 月 1 日以降に出荷された SPARC デスクトップモデルは、米国環境保護局 (EPA) の Energy Star Memorandum of Understanding #3 ガイドラインに準拠し、autoshutdown はデフォルトで無効になり、30 分間アイドル状態が続くと autopm が有効になります。これは、これらのマシンでデフォルトのキーワード autopm エントリが enabled になったと解釈されるためです。このデフォルト設定を確認するメッセージは表示されません。

EPA の Energy Star Memorandum で使用中のマシンで使用可能なバージョンを知るには、以下のようにします。

```
prtconf -pv | grep -i energystar
```

このプロパティがない場合は、使用可能な Energy Star のガイドラインがないということです。

システム電源管理 (保存停止・復元再開機能) は、現在限られたハードウェアプラットフォームでしかサポートされていません。システム電源管理 をサポートするプラットフォームは、『電源管理システム ユーザーマニュアル』に一覧表示されています。使用中のマシンでプログラミングに保存停止・復元再開機能が使用できるかどうかを判断するには、uname(2) を参照してください。