IRIX® Admin: Resource Administration (日本語版)

007–3700–002JP

制作スタッフ

Terry Schultz (執筆)

Rick Thompson (編集)

Chris Wengelski (イラスト)

Susan Gorski (製作)

Tom Goozen、Dennis Parker、Dan Stekloff (技術協力)

改訂情報

バージョン	説明
001	1999 年 7 月 初版
002	2000 年 1 月 IRIX リリース 6.5.7 をサポート

目次

図一覧

表一覧

このマニュアルについて

このマニュアルでは、SGI サーバ・システム上で動作する *IRIX 6.5.7* オペレーティング・システムについて説明しています。

このガイドは、IRIX オペレーティング・システムが動作している SGI コンピュータ・システムの管理者向けのリファレンス・ドキュメントです。多様なシステム・リソース管理機能の利用に必要な情報が含まれています。

このマニュアルには、以下の章が含まれます。

関連マニュアル

このガイドは、IRIX Admin のマニュアル・セットの一部であり、管理者向けの内容になっています。管理者とは、サーバ、複合システム、およびユーザのホーム・ディレクトリや作業ディレクトリ以下にあるもの以外のファイル構造に対して責任を持つ人のことです。他のユーザのためにシステムを保守する場合や、IRIX についてエンドユーザ・マニュアルよりも詳しい情報が必要な場合は、これらのガイドが役に立ちます。IRIX Admin ガイドは、IRIS InSight オンライン参照システムを通じて利用できます。このセットは、以下のマニュアルから構成されます。

- 『IRIX Admin: Software Installation and Licensing』- IRIX (SGI による UNIX オペレーティング・システム) における、ソフトウェアのインストールおよびライセンスの方法について説明します。このマニュアルには、ミニルートの実行方法や `inst`(1M) (IRIX インストール・ユーティリティに対するコマンドライン・インタフェース) を使用したライブ・インストールの説明が含まれています。また、IRIX で動作する特定のアプリケーションの利用を制御するライセンス管理製品について説明し、ライセンス製品のドキュメントの参照先を示します。
- 『IRIX Admin: System Configuration and Operation』一般的なシステム管理作業をリストアップし、システム管理タスクについて説明します。オペレーティング・システムの設定方法、ユーザ・

アカウント / ユーザ・プロセス / ディスク・リソースの管理方法、PROM モニタ内でのシステムとの対話方法、システム・パフォーマンスのチューニング方法が含まれます。

- 『IRIX Admin: Disks and Filesystems』ディスク、ファイルシステム、論理ボリュームの概念について説明します。SCSI ディスク、XFS や EFS (Extent File System) のファイルシステム、XLV 論理ボリューム、帯域保証 I/O に関するシステム管理手順について説明します。
- 『IRIX Admin: Networking and Mail』ネットワークとメール・システムの計画、設定、使用の方法について説明します。これには、sendmail、UUCP、SLIP、PPP の解説が含まれています。
- 『IRIX Admin: Backup, Security and Accounting』ファイルのバックアップと復元の方法、システムとネットワークのセキュリティを確保する方法、ユーザ毎のシステム使用状況を追跡する方法について説明します。
- 『IRIX Admin: Resource Administration』システム・リソース管理の手引きを提供し、さまざまな IRIX リソース管理機能 (IRIX のジョブ制限や Miser など) の使用方法と管理方法について説明します。
- 『IRIX Admin: Peripheral Devices』端末、モデム、プリンタ、CD-ROM ドライブやテープ・ドライブなどの周辺機器用のソフトウェアを設定、保守する方法について説明します。
- 『IRIX Admin: Selected Reference Pages』(InSight にはありません) システムダウン時に必要になる可能性のあるコマンドの使い方について、マン・ページの情報を簡潔に紹介しています。通常、個々のマン・ページは 1 つのコマンドを対象としていますが、密接な関係を持つ複数のコマンドを取り扱うマン・ページもあります。マン・ページは、`man`(1) コマンドを使用してオンラインで参照できます。

出版物の入手方法

SGI のドキュメントを入手するには、SGI Technical Publications Library (`http://techpubs.sgi.com`) にアクセスしてください。

表記上の決まり

このマニュアル全体を通して、次の表記規則を使用します。

表記規則	意味
`command`	この等幅フォントは、コマンド、ファイル、ルーチン、パス名、シグナル、メッセージ、プログラム言語の構造など、リテラル項目を示します。

variable	イタリック体は、変数のエントリや、定義される単語や概念を示します。
user input	このボールド体の等幅フォントは、対話型セッションでユーザが入力するリテラル項目を示します。出力は、ボールド体ではない等幅フォントで示されます。
[ ]	コマンドや命令文の省略可能部分は、角括弧で囲みます。
...	省略記号は、先行する要素が繰り返し可能であることを示します。

ご意見とお問い合わせ先

技術的正確性、内容、マニュアルの構成についてご意見がございましたら、弊社までお問い合わせください。必ず、コメントいただくマニュアルのタイトルとドキュメント番号もあわせてご連絡下さるよう、お願いいたします。(オンラインの場合、ドキュメント番号はマニュアルの最初の部分にあります。印刷マニュアルの場合は、裏表紙にドキュメント番号が記載されています。)

ご連絡は、以下のいずれかの方法をご利用いただけます。

- 電子メールの場合は、

  `techpubs@sgi.com`

  までお送りください。

- 次の Technical Publications Library の Web ページからの場合は、[Feedback] オプションをクリックしてください。

  `http://techpubs.sgi.com`

- お客様相談窓口までご連絡いただき、SGI 問題追跡システムへの入力をお申しつけください。

- 郵送の場合は、次の住所までお送りください。
  Technical Publications
  SGI
  1600 Amphitheatre Pkwy., M/S 535
  Mountain View, California 94043–1351, USA

- FAX の場合は、+1 650 932 0801 の“ Technical Publications” あてにお送りください。

いただいたコメントには迅速確実に対応いたします。

プロセス制限

標準のシステム・リソース制限を設定することにより、プロセスの作成時にプロセス・ベースの同じ制限を各ログイン・プロセスに適用できます。この章では、プロセス制限について説明しており、以下の節が含まれています。

プロセス制限の概要

IRIX オペレーティング・システムは、プロセスごとの制限をサポートします。プロセスやそれにより作成される各プロセスによる、さまざまなシステム・リソース消費量の制限は、`getrlimit`(2) システム・コールで取得し、`setrlimit`(2) システム・コールで設定することができます。

`getrlimit` または `setrlimit` の呼び出しでは、操作の対象となるリソースと、リソースの制限を指定します。リソースの制限は、一対の値です。1 つは、現在の (ソフト) 制限を指定し、もう 1 つは、最大 (ハード) 制限を指定します。プロセスは、ソフト制限をハード制限以下の任意の値に変更できます。プロセスは、ハード制限をソフト制限以上の任意の値に低くすることができます (一度低くしたハード制限を再び増やすことはできません)。

csh と sh を使用したリソース消費量の制限

`csh` (または `sh`) で `limit -h` *resource max-use* というコマンドを使用すると、現在のプロセス、またはその子プロセスによって消費されるリソースを制限できます。

このコマンドは、現在のプロセスとその各子プロセスが消費するリソースの総量を制限します。最大使用量を指定しないと、現在の制限が表示されます。リソースを指定しないと、すべての制限が示されます。`-h` フラグを指定すると、現在の制限の代わりに、ハード (最大) 制限が使用されます。ハード制限は、現在の制限に対する上限値を示します。最大 (ハード) 制限を大きくするには、`CAP_PROC_MGT` capability が必要です。

詳細については、csh(1) および sh(1) のマン・ページを参照してください。プロセスの特権に対するきめ細かな調整を提供する capability 機構の詳細については、capability(4) および capabilities(4) のマン・ページを参照してください。

systune を使用したプロセス制限の表示と設定

IRIX オペレーティング・システムによってサポートされるプロセス制限を表1-1 に示します。

表1-1 プロセス制限

制限名	シンボリック ID	単位	説明	違反時の動作
rlimit_cpu_cur rlimit_cpu_max	RLIMIT_CPU	秒	プロセスが使用できる CPU 時間 (秒) の最大値	SIGXCPU シグナルによるプロセスの終了
rlimit_fsize_cur rlimit_fsize_max	RLIMIT_FSIZE	バイト	プロセスが作成できるファイルの最大サイズ	上書きや追加が失敗し、errno に EFBIG が設定される
rlimit_data_cur rlimit_data_max	RLIMIT_DATA	バイト	プロセスの最大ヒープ・サイズ	brk(2) 呼び出しが失敗し、errno に ENOMEM が設定される
rlimit_stack_cur rlimit_stack_max	RLIMIT_STACK	バイト	プロセスの最大スタック・サイズ	SIGSEGV シグナルによるプロセスの終了
rlimit_ core_cur rlimit_core_max	RLIMIT_CORE	バイト	プロセスが作成できるコア・ファイルの最大サイズ	最大値に到達した時点でコア・ファイルの書き込みを終了
rlimit_nofile_cur rlimit_nofile_max	RLIMIT_NOFILE	ファイル記述子数	プロセスが同時に開くことができるできる、開いているファイル記述子の最大数	open(2) の試行が失敗し、errno に EMFILE が設定される

制限名	シンボリック ID	単位	説明	違反時の動作
rlimit_vmem_cur rlimit_vmem_max	RLIMIT_VMEM	バイト	プロセスの最大アドレス空間	brk(2) と mmap(2) の呼び出しが失敗し、errno に ENOMEM が設定される
rlimit_rss_cur rlimit_rss_max	RLIMIT_RSS	バイト	プロセスの常駐セット・サイズの最大サイズ	制限を超えた常駐ページが、最初のスワップ候補になる
rlimit_pthread_cur rlimit_pthread_max	RLIMIT_PTHREAD	スレッド数	プロセスが作成できるスレッドの最大数	スレッド作成が失敗し、errno に EAGAIN が設定される。

systune *resource* コマンドを使用して、プロセス制限のシステム・デフォルト値の表示と設定が行えます。*resource* には、以下の値が使えます。

```
rlimit_cpu_cur
rlimit_cpu_max
rlimit_fsize_cur
rlimit_fsize_max
rlimit_data_cur
rlimit_data_max
rlimit_stack_cur
rlimit_stack_max
rlimit_core_cur
rlimit_core_max
rlimit_nofile_cur
rlimit_nofile_max
rlimit_vmem_cur
rlimit_vmem_max
rlimit_rss_cur
rlimit_rss_max
rlimit_pthread_cur
rlimit_pthread_max
```

詳細については、systune(1M) のマン・ページを参照してください。

ジョブ制限ソフトウェアをシステムにインストールし、実行している場合は、ユーザ制限データベース (ULDB) にユーザベースのプロセス制限を設定することもできます。rlimit_cpu_cur と rlimit_cpu_max

のように、現在値と最大値の両方を指定できます。ULDB の値は、systune(1M) コマンドによって設定されるシステムのデフォルトよりも優先されます。

ULDB の詳細については、10 ページの「ユーザ制限データベース」を参照してください。

追加のプロセス制限パラメータ

IRIX には、特定のシステム制限を設定するためのパラメータが用意されています。たとえば、各プロセス (core やファイルのサイズ)、ユーザあたりのグループの数、常駐ページの数などの最大値を設定できます。以下に、maxup と cpulimit_gracetime について説明します。パラメータはすべて、/var/sysgen/mtune で設定、定義されています。

maxup	ユーザあたりのプロセスの最大数
cpulimit_gracetime	プロセスやジョブの制限の猶予期間

maxup パラメータやその他の“ システム制限パラメータ“ の詳細については、『IRIX Admin: System Configuration and Operation』を参照してください。

cpulimit_gracetime パラメータには、CPU 時間の制限を越えたプロセスの猶予期間を指定します。このパラメータには、制限を超えた後にプロセスの実行が許される秒数を設定します。cputlimit_gracetime が設定されない (0 (ゼロ) に設定されている) 場合は、プロセスかジョブの CPU 制限のどちらかを超えたプロセスに、SIGXCPU シグナルが送信されることになります。そのプロセスが実行を続ける限り、カーネルは SIGXCPU シグナルを周期的に送信します。プロセスは、SIGXCPU シグナルを独自に処理するように登録できるため、CPU 制限を無視することがあります。

systune(1M) コマンドを使用して、cpulimit_gracetime パラメータを 0 以外の値に設定すると、system は異なる動作を取ります。プロセスが CPU 制限を超えると、カーネルは SIGXCPU シグナルを 1 回だけプロセスに送信します。プロセスはこのシグナルを登録して、そこで必要なクリーンアップやシャットダウンの操作を行うことができます。cpulimit_gracetime で設定された CPU 猶予時間が経過してもなおプロセスが実行中の場合、カーネルは SIGKILL シグナルを使用してそのプロセスを終了させます。

ジョブ制限

標準のシステム・リソース制限を設定することにより、プロセス作成時にプロセス・ベースの同じ制限を各プロセスに適用することができます。プロセスを個々に制限する方法は便利ですが、個々のユーザが使用するリソースを任意に制限できるわけではありません。IRIX カーネルのジョブ制限機能では、特定のログイン・セッションやバッチ実行に関連する全プロセスを、「ジョブ」と呼ばれる単一の論理単位にカプセル化します。ジョブは、プロセスをログイン・セッション毎にグループ化するのに使用されるコンテナです。リソースの使用制限はユーザ毎に個々のジョブに対して適用され、その制限はカーネルによって強制適用されます。すべてのプロセスは特定のジョブに関連付けられ、一意のジョブ識別子 (ジョブ ID) によって識別されます。特定のジョブに属する複数のプロセスは、1 つの単位として制限、制御、照会、アカウンティングを適用できます。これにより、システム管理者は、CPU 時間、メモリ、ファイル容量、その他のシステム・リソースに対してジョブ固有の制限を設定することができます。ユーザ制限データベース (ULDB) を使用すれば、ユーザ毎に個別のジョブ制限が適用できます。ULDB が未定義の場合は、ジョブ制限は全ジョブに対して適用されます。ジョブ制限ソフトウェアを使用することで、マルチユーザ環境の大規模システムの利用が非常に容易になります。

この章には以下の節が含まれます。

最初にお読みください

この章の各節では、ジョブ制限ソフトウェアを、お使いのシステムにインストールする方法について説明しています。以下のリストに従って参照してください。

1. ジョブとジョブ制限に関する一般的な説明については、6 ページの「ジョブ制限の概要」と 8 ページの「サポートされるジョブ制限」を参照してください。

2. ジョブ制限パッケージのインストールについては、21 ページの「ジョブ制限のインストール」を参照してください。

3. ユーザ制限命令入力ファイル *infile* の記述と、ユーザ制限データベース (ULDB) の作成については、それぞれ12 ページの「ユーザ制限命令入力ファイルの作成」と11 ページの「ユーザ制限データベースの作成」を参照してください。

   ジョブ制限に関連するマン・ページの一覧は、47 ページの「マン・ページ」を参照してください。

4. `systune` *joblimits* コマンドを使用して、システム全体のジョブ制限のデフォルト値を設定する方法については、15 ページの「`systune` を使用したジョブ制限の表示と設定」を参照してください。

5. システムのジョブ制限を表示する方法については、16 ページの「ジョブ制限の表示・設定用ユーザ・コマンド」を参照してください。

6. ジョブ制限のインストールに関するトラブルシューティングについては、22 ページの「ジョブ制限のトラブルシューティング」を参照してください。

7. アプリケーション・プログラミング・インターフェースについては、22 ページの「ジョブ制限のアプリケーション・プログラミング・インタフェース」と24 ページの「ULDB のアプリケーション・プログラミング・インタフェース」を参照してください。

ジョブ制限の概要

ジョブ制限ソフトウェアを使用すると、各ユーザが適切な量のシステム・リソース (CPU 時間やメモリなど) にアクセスでき、それぞれの割り当て量を超えないようにするといった設定が容易に行えます。ジョブ制限ソフトウェアで各ユーザのマシン使用量を制限することにより、システムのスループットや利用率を高めることができます。IRIX でサポートされているユーザ・ベースのジョブ制限については、8 ページの「サポートされるジョブ制限」を参照してください。

システムの処理はさまざまな方法で起動されます。たとえば、端末からのログイン、ワークロード管理システムからの実行、`cron` ジョブ、または、`rsh`、`rcp`、`ftp`、array サービスといったリモート・アクセスなどです。このような各エントリ・ポイントは、元となるシェル・プロセスを作成し、そこから分かれる複数のプロセスを作成します。現在 IRIX オペレーティング・システムでは、個々のプロセスが使用可能なシステム・リソース数の制限がサポートされていますが、エントリ・ポイント・プロセスから生じたプロセスの総計に対する制限はサポートされていません。カーネル・ジョブは、エントリ・ポイントで生じるすべてのプロセスのリソース使用を制限する方法を提供します。ジョブはエントリ・ポイントのプロセスを祖先とするすべての関連プロセスのグループで、一意なジョブ ID によって識別されます。7 ページの図2-1は、ジョブの基点となるエントリ・ポイント・プロセスを示しています。

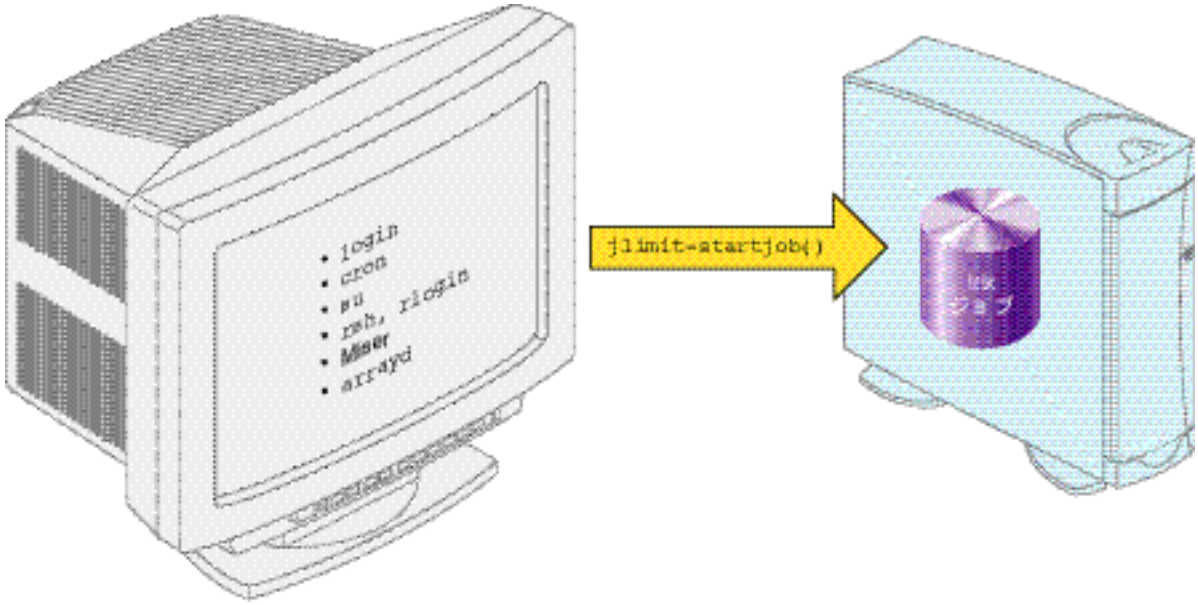

図2-1 エントリ・ポイント・プロセス

IRIX のジョブ制限には次のような特徴があります。

- ジョブはプロセスを囲い込むためのコンテナである。プロセスは、ジョブの外部に出たり、明示的なアクション (ルート権限でのシステム・コール) なしにジョブの外部に新しいジョブを作成できない。
- 各新規プロセスは、親プロセスからジョブ ID と制限を継承する。
- すべてのエントリ・ポイント・プロセス (ジョブ・イニシエータ) は、新規ジョブを作成し、該当するジョブ制限を設定する。
- ユーザは、自身のジョブ制限をシステム管理者によって指定された最大値の範囲内で増減させることが可能。
- ジョブ・イニシエータが認証とセキュリティ・チェックを行う。

プロセス制御初期化プロセス (`init`(1M)) と init によって呼び出されるスタートアップ・スクリプトは、ジョブの一部ではなく、ジョブ ID は 0 (ゼロ) になります。

Network Queuing Environment (NQE) と Load Sharing Facility (LSF) は、このリリースのジョブ制限ソフトウェアではサポートされていません。

メモ: 既存の IRIX コマンドである jobs(1)、fg(1)、bg(1) のマン・ページはシェルの“ ジョブ” に関するもので、IRIX カーネルのジョブとは関係ありません。

図2-2に 2 つの制限ドメインを示します。制限ドメインは作業を分類するための 1 つの手法です。ジョブ・イニシエータ (7 ページの図2-1参照) は、対話プロセスまたはバッチ・プロセスに分類できます。制限ドメイン名は、ユーザ制限データベース (ULDB) 作成時にシステム管理者によって定義されます。ULDB を使用してジョブ制限情報を取得するアプリケーションは、名前の指定によって制限情報が検索できることを想定しています。この名前は規則に基づいて定義されます。制限ドメインとULDB の詳細については、10 ページの「ユーザ制限データベース」を参照してください。

図2-2 制限ドメイン

サポートされるジョブ制限

IRIX オペレーティング・システムによってサポートされるジョブ制限を 9 ページの表2-1 に示します。各制限項目を使用して、ジョブ内の全プロセスによる特定のシステム・リソースの使用総量を制限します。ジョブ制限ソフトウェアには、各ジョブ内のプロセス数を制御する JLIMIT_NUMPROC と呼ばれる制限も用意されています。

表2-1 ジョブ制限

制限名	シンボリック ID	単位	説明	違反時の動作
jlimit_nproc_cur jlimit_nproc_max	JLIMIT_NUMPROC	プロセス	ジョブ内のプロセスの最大数	ジョブによるプロセス生成は失敗し、errno に EAGAIN が設定される
jlimit_nofile_cur jlimit_nofile_max	JLIMIT_NOFILE	ファイル記述子	ジョブ内のすべてのプロセスが保持できる、開いているファイル記述子の合計の最大数	ジョブによる open(2) の呼び出しが失敗し、errno に EMFILE が設定される
jlimit_rss_cur jlimit_rss_max	JLIMIT_RSS	バイト	ジョブ内の全プロセスに対する常駐セット・サイズの合計の最大値	制限を超えた常駐ページが、最初のスワップ候補になる
jlimit_vmem_cur jlimit_vmem_max	JLIMIT_VMEM	バイト	ジョブ内の全プロセスに対するアドレス空間の合計の最大値	ジョブでの brk(2) と mmap(2) の呼び出しが失敗し、errno に ENOMEM が設定される
jlimit_data_cur jlimit_data_max	JLIMIT_DATA	バイト	ジョブ内の全プロセスに対するヒープ・サイズの合計の最大値	ジョブによる brk(2) の呼び出しが失敗し、errno に ENOMEM が設定される
jlimit_cpu_cur jlimit_cpu_max	JLIMIT_CPU	秒	ジョブ内の全プロセスに許可される CPU 時間 (秒) の合計の最大値	SIGXCPU シグナルによるジョブ内の全プロセスの終了

ジョブのシステム・リソース消費に関する制限 (9 ページの表2-1参照) は、getjlimit(2) 関数で取得し、setjlimit(2) 関数で設定できます。getjlimit 関数は、指定したジョブのジョブ制限の現在の値と最大値を取得します。他のユーザが所有するジョブの値を取得するには、CAP_MAC_READ capability が必要です。

setjlimit(2) 関数は、指定したジョブのジョブ制限の現在値と最大値を設定します。現在のジョブが設定しようとするジョブと異なる場合は、setjlimit 関数は CAP_MAC_WRITE capability をチェックします。最大 (ハード) 制限を増加しようとしている場合は、setjlimit 関数は CAP_PROC_MGT capability をチェックします。

詳細については、getjlimit(2) のマン・ページを参照してください。プロセスの特権に対する細かな調整を可能にする capability の詳細は、capability(4) および capabilities(4) のマン・ページを参照してください。

systune *joblimits* コマンドを使用して、システム全体のデフォルト値の設定が行えます。詳細については、15 ページの「systune を使用したジョブ制限の表示と設定」と systune(1M) のマン・ページを参照してください。

cpulimit_gracetime パラメータで、CPU 時間の制限を超えたプロセスの猶予期間を指定します。cpulimit_gracetime パラメータの詳細については、4 ページの「追加のプロセス制限パラメータ」を参照してください。

ジョブ制限ソフトウェアの動作は、cpulimit_gracetime パラメータ・ソフトウェアに似ています。プロセスが実行されるにつれて CPU 利用量が増加します。制限値に達すると、SIGXCPU シグナルが各プロセスに対してプロセス実行時に個別に送信されます。SIGXCPU がプロセスに送信されると、そのプロセスで猶予期間が有効になります。猶予期間が期限切れになった時点でまだプロセスが実行中である場合、プロセスは SIGKILL シグナルによって終了させられます。ジョブ内で実行されているプロセスに対してのみ SIGXCPU シグナルが送信されます。ジョブ内の各プロセスは個別の猶予期間を取得します。そのため、SIGXCPU シグナルをジョブ内の全プロセスにまとめて送信することはできません。

ユーザ制限データベース

ユーザ制限データベース (ULDB) にはジョブ制限の情報が含まれており、この情報を使ってシステム管理者はユーザ単位でマシンへのアクセスを制御することができます。ジョブ・イニシエータ、つまり、新規のジョブをシステム上で起動するアプリケーション (login、 rsh、rlogin、cron、ftp、また、Miser などのワークロード管理システムなど) は、特定のユーザのジョブ制限値を ULDB から取得し、その情報を使用して該当する制限を設定します。

ジョブ・イニシエータに関する詳細は、6 ページの「ジョブ制限の概要」を参照してください。

ULDB は、ジョブ制限パッケージをインストールする際に、ジョブのジョブ制限値とプロセス制限値の設定に使用されます。ジョブ制限をインストールしない場合、現行のリソース制限機能によりプロセス制限が行われます。

あるユーザに対して、値を記述した「user」エントリがない場合は、ドメインのデフォルトがそのユーザに適用されます。ユーザ毎の値は、ドメインのデフォルト値より優先されます。ULDB の値は、ジョブ制限とプロセス制限の両方のシステム・デフォルト値より優先されます。

この節では、ULDB の内容の作成、メンテナンス、表示に使用するコマンドや、アプリケーションから ULDB 情報にアクセスするためのライブラリのアプリケーション・プログラミング・インターフェース (API) について説明しています。

メモ: /etc/jlimits.in ファイルに含まれている ULDB 設定ファイルには、ULDB をセットアップする際に利用できるテンプレートが含まれています。

/etc ディレクトリには、jlimits と jlimits.m ファイルも含まれています。jlimits.in ファイルは、ジョブ制限をローカル ULDB の jlimits.m ファイルや NIS のマスター・マップに読み込む際に使用されるコロン区切りの jlimits ファイルにパースされます。jlimits ファイルは genlimits(1M) コマンドによって自動的に生成されます。jlimits.m ファイルはローカルの ULDB の mdbm ファイルです。

ユーザ制限データベースの作成

ULDB を作成するコマンドを次に示します。

genlimits [-i *infile*] [-l] [-m] [-L *local_database*] [-N *nisfile*] [-v]

genlimits コマンドは、フォーマットされた ASCII ユーザ制限命令入力ファイル (*infile*) をコロン区切りの ASCII ファイルに変換し、以下のいずれかの形式に出力します。

- Network Information Service (NIS) サーバ・マップ (-m オプション)
- NIS 用または直接使用 (非 NIS用) のローカル・データベース (-l オプション)

genlimits コマンドでは次のオプションが使用できます。

-i *infile*	ユーザ制限命令入力ファイルの場所を指定する。-i オプションが指定されない場合のデフォルトファイルは /etc/jlimits.in になります。
-l	Network Information Service (NIS) 用、または、直接使用する場合の非 NIS ローカル・データベースを作成する。NIS が使用可能な場合、ローカル・データベースにはローカルのエントリが含まれ、それにより NIS サーバのエントリに対して上書きや追加が行われます。NIS が使用可能でない場合、ローカル・データベースにはシステムでの制限を設定する情報が含まれます。デフォルトでは、このデータベースは /etc/jlimits.m ファイルに含まれています。-l オプションは、-m オプションと同時には使用できません。
-m	NIS マスター・サーバ・マップを作成する。標準の NIS マップの保存場所にマップを生成し、保存します。NIS マップファイルの保存場所は変更できません。-m オプションは、-l オプションと同時には使用できません。
-L *local_database*	ローカル・データベースの保存場所を指定する。-L オプションは -l オプションと組み合わせて使用します。

-N *nisfile*	NIS データベースの作成済みソース入力ファイルの保存場所を指定する。デフォルトの保存場所は、/etc/jlimits です。-N *nisfile* オプションを使用すれば、既存の /etc/jlimits ファイルを上書きせずにデータベースを新規作成できます。
-v	genlimits コマンドの動作を示すメッセージを出力する冗長モードを指定します。

詳細については、genlimits(1M) のマン・ページを参照してください。

ユーザ制限命令入力ファイルの作成

ユーザ制限命令ファイルは、ULDB 生成に使用するドメイン、制限、ユーザの情報を定義しており、genlimits(1M) コマンドへの入力に使用されます。この節では、ユーザ制限命令入力ファイルの記述方法について説明しています。

コメント

コメント記号 (#) に続くテキストはすべてコメントとして扱われます。

数値の制限値

以下のように、数値に文字を追加して、制限値を決定する数値に適用する乗数を表すことができます。

文字	乗数値
k (キロ)	1024 (2**10)
m (メガ)	1,048,576 (2**20)
g (ギガ)	1,073,741,824 (2**30)
t (テラ)	1,099,511,627,776 (2**40)
H (時)	3600
M (分)	60

- k、m、g、t の乗数はメモリの制限やその他の大きな値を定義する際に使用します。
- H と M の乗数は時間の値を定義する際に使用します。

乗数値は、システム・インクルード・ファイル /usr/include/uldb.h に定義されています。

上記の方法で乗数を使用する場合の前提条件はありません。

ある特定の制限タイプに対しては、数値の制限値を指定するかわりに上限がないことを示す“ unlimited”を指定することもできます。

ULDB 作成の詳細については、genlimits(1M) の マン・ページを参照してください。

Domain 命令

ULDB で定義される各制限ドメインは、命令「domain」で始めなければなりません。この命令には、ASCII 文字で記述されたドメイン名と、ドメインに対するデフォルトの制限値のリストを指定します。domain 命令の例を次に示します。

```
domain domain_name {
      limit_name = value
      limit_name:machname = value
      ...
   }
```

ドメイン interactive および batch は、ユーザ・ジョブ制限用に予約されています。その他のドメイン名は、特定用途のために作成、使用が可能です。

User 命令

「user」命令は、個々のユーザに対する制限を設定します。ユーザ名には有効なログイン・アカウントを指定する必要があります。uid 値は省略可能です。uid が指定されている場合、genlimits コマンドによって、指定された uid が genlimits を実行したマシンのユーザの uid と一致するかどうか検証されます。domain 節では、ユーザが一意の制限値を持つ各ドメインを指定します。user 命令にリストされるドメインは、それ以前に記述されている domain 命令ですでに定義されている必要があります。domain 節の構造体とその意味は、domain 命令と同じです。システムの全ユーザに対して user 命令を記述する必要はありません。照会したユーザに対する user 命令がない場合や、照会したドメインに対する値がない場合、そのドメインに対するデフォルト値が返されます。user 命令の例を次に示します。

```
user user_name[:uid] {
      domain_name {
         limit_name = value
         limit_name:machname = value
         ...
      }
      domain_name {
         ...
      }
      ...
   }
```

domain 命令と user 命令の制限指定には、マシン名を含めることが可能です (省略可能)。マシン名を含めて指定した制限値は、該当するマシンに対してのみ適用されます。マシン名を指定しない制限は、

クラスタ内のすべてのマシンに適用されます。ひとつの命令入力ファイルにはマシン名を指定しない制限に加えて、それと同じ制限を異なるマシン名の指定付きでくり返し記述できます。

genlimits コマンドは、下記のルールに従って、生成されるデータベースのタイプ (11 ページの「ユーザ制限データベースの作成」参照) に応じて、マシン名に関連付けられた制限値を処理します。

- -m オプションを使用して NIS マスター・マップを生成した場合、マシン名に関連付けられている制限値は無視されます。マシン名を指定していない、クラスタ全体に適用される値だけがデータベースに含まれることになります。
- -l オプションを使用してローカル・データベースを生成した場合、genlimits コマンドは、ローカル・マシン名に関連付けられている制限値を (存在する場合は) 選択します。ローカル・マシン名に関連付けられている制限値がない場合、genlimits コマンドは、マシン名と関連付けられていない、クラスタ全体に適用される値を選択します。ローカル・マシン名を調べるには、uname -n コマンドを実行します。uname コマンドの詳細については、uname(1) のマン・ページを参照してください。

ユーザ制限命令入力ファイルのセットアップの例

genlimits コマンドを実行するたびに、ULDB は完全に再構築されるため、入力命令ファイルにはデータベースを完全に作成できる情報が含まれている必要があります。情報の変更が必要な場合、システム管理者はユーザ制限命令入力ファイルを編集し、データベースを再構築する必要があります。特定のユーザに対する user エントリがない場合はドメインのデフォルトが使用されるため、管理者は、必要なユーザに対してのみ名前を指定した user エントリを作成してデフォルト値を上書きする必要があります。次に、3 つの制限タイプ、2 つのドメイン、個別の制限のある 1 ユーザを指定した、ユーザ制限命令入力ファイルの例を示します。制限およびドメイン名は、この例でのみ使用されるもので、実際の制限値を反映したものではありません。ULDB には値のみが保存されます。値とその単位の意味は、制限を使用するアプリケーションに依存します。

メモ: ULDB のエントリを更新している際にお使いのシステムにおけるジョブ制限値が変更されない場合、ULDB 内で使用される制限名と systune *joblimits* グループで使用される制限名を必ず同じにしてください。詳細については、22 ページの「ジョブ制限のトラブルシューティング」を参照してください。

```
domain interactive {              # domain for interactive logins
   jlimit_cpu_cur = 60
   jlimit_cpu_max = 120           # limit interactive jobs to 120 CPU seconds
   jlimit_vmem_cur = 2m
   jlimit_vmem_max = 4m           # limit interactive jobs to 4 megabytes of virtual memory
   jlimit_numproc_cur =10
   jlimit_numproc_max = 20        # limit interactive jobs to 20 concurrent processes
}
```

```
domain batch {                      # domain for batch submissions
   jlimit_cpu_cur = 3600
   jlimit_cpu_max = 7200            # limit batch jobs to two hours of CPU time
   jlimit_vmem_cur = 128m
   jlimit_vmem_max = 256m           # limit batch jobs to 256 megabytes of memory
   jlimit_numproc_cur = unlimited
   jlimit_numproc_max = unlimited # no limit on processes in a batch job
}

user fred:123 {                     # User "fred" gets his own interactive CPU limits
   interactive  {                   #
      jlimit_cpu_cur = 300
      jlimit_cpu_max = 600          # "fred" needs to run longer jobs in interactive mode
    }
}
```

systune を使用したジョブ制限の表示と設定

systune *joblimits* コマンドを使用して、システム全体に及ぶプロセス制限のデフォルト値の表示と設定が行えます。ULDB が存在する場合は、ULDB の値はこれらの値より優先されます。*joblimits* グループには、以下の変数があります。

```
jlimit_cpu_cur
jlimit_cpu_max
jlimit_data_cur
jlimit_data_max
jlimit_vmem_cur
jlimit_vmem_max
jlimit_rss_cur
jlimit_rss_max
jlimit_nofile_cur
jlimit_nofile_max
jlimit_numproc_cur
jlimit_numproc_max
```

systune *joblimits* コマンドからの出力を次に示します。

```
$ systune joblimits
group: joblimits (statically changeable)
        jlimit_numproc_max = 1024 (0x400) ll
        jlimit_numproc_cur = 1024 (0x400) ll
```

```
jlimit_nofile_max = 5000 (0x1388) ll
jlimit_nofile_cur = 400 (0x190) ll
jlimit_rss_max = 9223372036854775807 (0x7fffffffffffffff) ll
jlimit_rss_cur = 9223372036854775807 (0x7fffffffffffffff) ll
jlimit_vmem_max = 9223372036854775807 (0x7fffffffffffffff) ll
jlimit_vmem_cur = 9223372036854775807 (0x7fffffffffffffff) ll
jlimit_data_max = 9223372036854775807 (0x7fffffffffffffff) ll
jlimit_data_cur = 9223372036854775807 (0x7fffffffffffffff) ll
jlimit_cpu_max = 9223372036854775807 (0x7fffffffffffffff) ll
jlimit_cpu_cur = 9223372036854775807 (0x7fffffffffffffff) ll
```

表示情報の内容は次の通りです。

- `jlimit_numproc` - プロセス制限の数
- `jlimit_nofile` - ファイル制限の数
- `jlimit_rss` - 常駐セット・サイズ。デフォルトはバイト単位
- `jlimit_vmem` - 仮想メモリの制限。デフォルトはバイト単位
- `jlimit_data` – データ・サイズ。デフォルトはバイト単位
- `jlimit_cpu` - CPU 時間。デフォルトは秒単位

詳細については、`systune`(1M) および `jlimit`(1) のマン・ページを参照してください。

ジョブ制限の表示・設定用ユーザ・コマンド

この節では、ジョブ制限の表示と設定に使用する以下のユーザ・コマンドについて説明しています。

- 16 ページの「`showlimits`」
- 19 ページの「`jlimit`」
- 20 ページの「`jstat`」

`showlimits`

ULDB の制限情報を表示するコマンドは、次のようになります。

`showlimits [-D] [-d] [-u` *user_name*`] [`*domain_name*`]`

`showlimits` コマンドはユーザ制限データベース (ULDB) の制限情報を表示します。

showlimits コマンドでは次のオプションが使用できます。

-D	ULDB 内に定義されたすべてのドメイン名を表示する。-D オプションが指定されている場合は、ドメイン名とその他のオプションは無視されます。
-d	ドメインのデフォルトの制限を表示する。オプションが指定されていない場合は、showlimits コマンドはすべてのドメインに対するデフォルトの制限を表示します。
-u *user_name*	現在のユーザの替わりに、指定されるユーザに対する制限値を表示する。
domain_name	すべてのドメインの替わりに、指定されるドメインに対する制限値を表示する。

オプションが指定されていない場合は、showlimits コマンドはすべてのドメインでの現在のユーザに対する現在の制限情報を次のように表示します。

```
% showlimits

Domain interactive:
        jlimit_cpu_cur: unlimited
        jlimit_cpu_max: unlimited
        jlimit_data_cur: unlimited
        jlimit_data_max: unlimited
        jlimit_nofile_cur: 400
        jlimit_nofile_max: unlimited
        jlimit_vmem_cur: unlimited
        jlimit_vmem_max: unlimited
        jlimit_rss_cur: unlimited
        jlimit_rss_max: unlimited
        jlimit_pthread_cur: 2k
        jlimit_pthread_max: 65535
        jlimit_numproc_cur: 1k
        jlimit_numproc_max: 65535
        rlimit_cpu_cur: unlimited
        rlimit_cpu_max: unlimited
        rlimit_fsize_cur: unlimited
        rlimit_fsize_max: unlimited
        rlimit_data_max: unlimited
        rlimit_stack_cur: 64m
        rlimit_stack_max: unlimited
        rlimit_core_cur: unlimited
        rlimit_core_max: unlimited
```

```
        rlimit_nofile_cur: 200
        rlimit_nofile_max: unlimited
        rlimit_vmem_max: unlimited
        rlimit_rss_max: unlimited

Domain batch:
        jlimit_cpu_cur: unlimited
        jlimit_cpu_max: unlimited
        jlimit_data_cur: unlimited
        jlimit_data_max: unlimited
        jlimit_nofile_cur: 400
        jlimit_nofile_max: unlimited
        jlimit_vmem_cur: unlimited
        jlimit_vmem_max: unlimited
        jlimit_rss_cur: unlimited
        jlimit_rss_max: unlimited
        jlimit_pthread_cur: 2k
        jlimit_pthread_max: 65535
        jlimit_numproc_cur: 1k
        jlimit_numproc_max: 65535
        rlimit_cpu_cur: unlimited
        rlimit_cpu_max: unlimited
        rlimit_fsize_cur: unlimited
        rlimit_fsize_max: unlimited
        rlimit_data_max: unlimited
        rlimit_stack_cur: 64m
        rlimit_stack_max: unlimited
        rlimit_core_cur: unlimited
        rlimit_core_max: unlimited
        rlimit_nofile_cur: 200
        rlimit_nofile_max: unlimited
        rlimit_vmem_max: unlimited
        rlimit_rss_max: unlimited
```

メモ: ユーザがログインした後で ULDB が変更された場合、現在の制限は有効にはなりません。現在の制限は新たにログインしたユーザに対して有効になります。

ジョブ制限値の説明については、9 ページの表2-1を参照してください。プロセス制限値の説明については、2 ページの表1-1を参照してください。

詳細については、showlimits(1) のマン・ページを参照してください。

jlimit

ジョブ制限の表示と設定を行うコマンドは、次のようになります。

jlimit [-j *job_id*] [-h] [*limit_name* [*value*]]

jlimit コマンドは、ジョブのリソース使用に関する制限の表示と変更を行います。ユーザに対して、ジョブの開始時にユーザ制限データベース (ULDB) 情報に含まれる値を使用して、現在の制限と最大 (ハード) 制限が設定されます。制限の最大値を超えない範囲で、現在の制限を増減させることが可能です。また、制限の最大値を減少させることはできますが、あとで元に戻すことはできません。制限の最大値を大きくするには、CAP_PROC_MGT Capability が必要です。使用している制限の最大値にかかわらず、現在の制限に達すると常に制限の違反時の動作が起こります。プロセスの特権に対する細かな調整を可能にする Capability の詳細は、capability(4) および capabilities(4) のマン・ページを参照してください。

jlimit コマンドでは次のオプションが使用できます。

-j *job_id*	制限に変更するジョブのジョブ ID を指定する。他のユーザに属するジョブのジョブ制限を変更するには、CAP_MAC_WRITE および CAP_PROC_MGT Capability が必要です。ジョブ ID は、16 進で出力されます。ジョブ ID を指定する場合、接頭辞「0x」は省略可能です。
-h	ジョブの最大 (ハード) 制限値の表示または変更を行うことを指定する。-h オプションが指定されていない場合は、jlimit コマンドは現在の制限値の表示または変更を行います。
limit_name[*value*]	指定した制限に対する値の設定と表示を行う。 • 制限名が指定されていない場合は、jlimit はすべての制限の値を表示します。 • 制限名が値なしで指定されている場合は、jlimit は制限の値を表示します。 • 制限名と値の両方が指定されている場合は、jlimit は該当する制限の値を設定します。

-j オプションと引数 *job_id* が指定されている場合、jlimit コマンドは次のような情報を出力します。

```
 % jlimit -j 0x14
cputime: unlimited
datasize: unlimited
files: unlimited
```

```
vmemory: unlimited
ressetsize: unlimited
processes: 65535
```

制限値の説明については、9 ページの表2-1を参照してください。

詳細については、jlimit(1) のマン・ページを参照してください。

jstat

アクティブなジョブのステータス情報を表示するコマンドは、次のようになります。

```
jstat [-a] [-l] [-p]
jstat [-j job_id] [-l] [-p]
```

jstat コマンドでは次のオプションが使用できます。

-a	すべてのジョブの情報を表示する。
-j *job_id*	指定したジョブ ID (*job_id*) の情報のみを表示する。
-l	現在のジョブまたは指定したジョブに関する制限情報 (現在の利用状況、現在の制限、制限の最大値など) を表示する。
-p	現在のジョブまたは指定したジョブに属する各プロセスの情報 (プロセス ID、状態、実行コマンドなど) を表示する。

-a または -j *job_id* のいずれも使用されない場合は、jstat コマンドは現在のジョブの情報を表示します。

-l オプションが指定されている場合は、jstat コマンドは、現在のジョブに関する現在の利用状況、最高利用時、現在の制限、制限の最大値の情報を次のように表示します。

```
% jstat -l

JID              OWNER          COMMAND
---------------- -------------- --------------
0x5eac0000001bd terry             -csh

LIMIT NAME       USAGE          HIGH USAGE     CURRENT LIMIT  MAX LIMIT
---------------- -------------- -------------- -------------- --------------
cputime          1:05           1:05           unlimited      unlimited
datasize         400k           400k           unlimited      unlimited
files            10             35             400            5000
vmemory          44             201            unlimited      unlimited
```

```
ressetsize      340             357             unlimited       unlimited
processes       2               4               1024            1024
```

集計情報は常に出力されます。制限値の説明については、9 ページの表2-1を参照してください。

詳細については、jstat(1) のマン・ページを参照してください。

ジョブ制限と既存の IRIX ソフトウェア

ps -j コマンドは、プロセス ID、プロセス・グループ ID、セッション ID、ジョブ ID を 16 進で出力します。

```
% ps -j
        PID       PGID        SID        JID TTY          TIME CMD
     253430     253430     253430     0x5eac001bd ttyq12  0:00 csh
     254563     254563     253430     0x5eac001bd ttyq12  0:00 ps
```

詳細については、ps(1) のマン・ページを参照してください。

クラスタ内の他のマシン上でジョブの新規プロセスが開始されると、array サービス・デーモン (arrayd(1M)) によってジョブ ID が起動元のマシンから他のマシンに伝えられます。

詳細については、arrayd(1M) のマン・ページを参照してください。

ジョブ制限のインストール

カーネル・ジョブ制限ソフトウェアをインストールするには、ソフトウェア・インストール・ツール inst(1M) または swmgr(1M) ソフトウェア管理ツールを使用します。inst(1M) と swmgr(1M) の詳細は、IRIX Admin マニュアル・セットの『IRIX Admin: Software Installation and Licensing』と該当するマン・ページを参照してください。

カーネル・ジョブ制限ソフトウェアを IRIX システムにインストールするには、次のサブ・システムをインストールしてください。eoe.sw.jlimits

ジョブ制限ソフトウェアをインストールした後、autoconfig(1M) コマンドを実行してシステムを再起動します。

ジョブ制限ソフトウェアは IRIX フィーチャ・ストリームでのみ利用可能です。

ジョブ制限のトラブルシューティング

ULDB のエントリを更新しても、システムにおけるジョブ制限値が変更されない場合、ULDB 内で使用される制限名と `systune` *joblimits* グループで使用される制限名が正確に一致しているかどうか確かめて下さい。ULDB からは、どのジョブ制限変数が有効でどれが無効なのかは決定できません。ULDB のシンボリック名の入力が不正確な場合、`systune` *joblimits* グループからの値が適用されます。制限名の詳細については、9 ページの表2-1 を参照してください。

ジョブ制限のアプリケーション・プログラミング・インタフェース

この節では、アプリケーション・プログラミング・インタフェース (API) 関数へのライブラリ・インタフェースが使用するデータ型と関数コールについて説明しています。

データ型

ここでは、API 関数へのライブラリ・インタフェースで使用される固有のデータ型について説明します。

すべての制限値は、`/usr/include/sys/resource.h` システム・インクルード・ファイル内でプロセス制限に対して定義された `rlimit` 構造体で指定されます。

```
typedef unsigned long rlim_t;
   struct rlimit_t {
           rlim_t         rlim_cur;
           rlim_t         rlim_max;
   };
```

ジョブ ID は、符号付き 64 bit 値として定義されます。これは、アプリケーションによって非透過的に処理されます。`jid_t` は、システム・インクルード・ファイル `sys/types.h` に定義されています。

```
typedef int64_t jid_t;
```

関数コール

ジョブ制限に対する API は、`libc.a` ライブラリで定義されている一連の関数によって定義されています。各関数は、`syssgi`(2) システム・インタフェースを呼び出して必要な操作を行います。関数のプロトタイプは、システム・インクルード・ファイル `/usr/include/sys/resource.h` にあります。

getjlimit および setjlimit

getjlimit 関数は各種のシステム・リソースの利用に関する制限をジョブ毎に取得し、setjlimit 関数はこれらの制限を設定します。

```
#include <sys/resource.h>
int getjlimit(jid_t jid, int resource, struct rlimit *rlp)
int setjlimit(jid_t jid, int resource, struct rlimit *rlp)
```

詳細については、getjlimit(2) のマン・ページを参照してください。

getjusage

getjusage 関数は、特定のジョブ ID のリソース利用値を取得します。

```
#include <sys/resource.h>
int getjusage(jid_t jid, int resource, struct jobrusage *up)
```

詳細については、getjusage(2) のマン・ページを参照してください。

getjid

getjid 関数は、現在のプロセスに関連付けられているジョブ ID を返します。

```
#include <sys/resource.h>
jid_t getjid(void);
```

詳細については、getjid(2) のマン・ページを参照してください。

killjob

killjob 関数は、指定したジョブ ID に属するすべてのプロセスにシグナルを送信します。

```
#include <sys/resource.h>
int killjob(jid_t jid, int signal)
```

詳細については、killjob(2) のマン・ページを参照してください。

makenewjob

makenewjob 関数は、新しいジョブ・コンテナを作成します。

```
#include <sys/resource.h>
jid_t makenewjob(uid_t user, jid_t rjid)
```

詳細については、makenewjob(2) のマン・ページを参照してください。

jlimit_startjob

jlimit_startjob 関数は、ジョブを新規作成し、ジョブ制限を ULDB 内の制限値に設定します。

jlimit_startjob の宣言を次に示します。

```
#include <sys/resource.h>
jid_t    jlimit_startjob(char *username, uid_t uid, char *domainname);
```

エラー・メッセージ

エラー・メッセージについては、該当するマン・ページか47 ページの付録 A「ジョブ制限に関するエラー・メッセージとマン・ページの概要」を参照してください。

ULDB のアプリケーション・プログラミング・インタフェース

この節では、ULDB へのライブラリ・インタフェースで使用されるデータ型と関数コールについて説明しています。

データ型

ここでは、ユーザ制限情報へのライブラリ・インタフェースで使用される固有のデータ型を定義します。ULDB 定義はすべてインクルード・ファイル /usr/include/uldb.h にあります。

2 進の制限値は、次のように符号なし 64 bit 値として保持されます。

```
typedef rlim_t uldb_limit_t;
```

uldb_namelist_t

uldb_namelist_t データ型は、制限名、ドメイン名などの名前リストの保持に使用します。namelist 構造体には、アイテムの個数と名前ポインタのリストへのポインタが含まれます。uldb_namelist_t データ型を次に示します。

```
typedef struct uldb_namelist_s {
      int uldb_nitems,                # number of names in the list
      char **uldb_names               # list of name pointers
   } uldb_namelist_t;
```

uldb_limitlist_t

uldb_limitlist_t データ型は、2 進の制限値のリストの保持に使用します。制限リスト構造体には、アイテムの個数と制限値の配列へのポインタが含まれます。uldb_limitlist_t データ型を次に示します。

```
typedef struct uldb_limitlist_s {
      int uldb_nitems,                # number of limit values in the list
      uldb_limit_t *uldb_limits       # list of limit pointers
   } uldb_limitlist_t;
```

関数コール

ここでは、ユーザ制限情報へのライブラリ・インタフェースで使用される関数コールを定義します。

制限値を取得する関数を以下に示します。

- uldb_get_limit_values
- uldb_get_limit_names
- uldb_get_domain_names

uldb_get_limit_values

uldb_get_limit_values 関数は、ドメインやユーザの制限値のセットの取得に使用します。指定したユーザに対する明示的なエントリがない場合は、ドメインのデフォルトを返します。要求した制限のセットは、uldb_namelist_t 構造体として取得されます。返された制限リストのポインタは、malloc ルーチン・コールで作成された新規の uldb_limitlist_t 構造体を参照します。不要になった構造体はアプリケーション側で開放する必要があります。返された uldb_limitlist_t 構造体に格納される制限値の順序は、入力した uldb_namelist_t 構造体の制限名の順序に対応します。ユーザ名が NULL の場合、ユーザ制限の代わりにドメインの制限のリストが返ります。

uldb_get_limit_values の例を次に示します。

```
#include include/uldb.h
uldb_limitlist_t *        # returns pointer to limit list or NULL if error
   uldb_get_limit_values (         #
      char *domain_name,           # pointer to domain name
      char *user_name,             # name of user
      uldb_namelist_t *limits);    # namelist containing limit names
```

uldb_get_value_units

uldb_get_value_units 関数は、指定した制限のリストの修飾値や単位を含む制限リスト構造体を返します。使用可能な修飾値は、ヘッダ・ファイル uldb.h に定義されています。返される名前リストは、malloc ルーチン・コールで作成された uldb_namelist_t 構造体に格納されます。不要になった構造体はアプリケーション側で開放する必要があります。

uldb_get_value_units の例を次に示します。

```
#include <include/uldb.h>
uldb_limitlist_t *        # returns pointer to limit list or NULL if error
   uldb_get_value_units (          #
      char *domain_name,           # pointer to domain name
      char *user_name,             # name of user
      uldb_namelist_t *limits);    # namelist containing limit names
```

uldb_get_limit_names

uldb_get_limit_names 関数は、ドメインに対して定義された制限名をすべて含んだリストの取得に使用します。返される名前リストは、malloc ルーチン・コールで作成された uldb_namelist_t 構造体に格納されます。不要になった構造体はアプリケーション側で開放する必要があります。

uldb_get_limit_names の例を次に示します。

```
#include <include/uldb.h>
   uldb_namelist_t *              # returns pointer to name list or NULL if error
   uldb_get_limit_names (
       char *domain_name);        # pointer to domain name
```

uldb_get_domain_names

uldb_get_domain_names 関数は、ULDB 内に定義されたドメイン名の完全なリストの取得に使用します。返される名前リストは、malloc ルーチンを呼び出して作成した uldb_namelist_t 構造体に格納されます。不要になった構造体はアプリケーション側で開放する必要があります。

```
#include <include/uldb.h>
uldb_namelist_t *               # returns pointer to name list or NULL if error
   uldb_get_domain_names (
       void);
```

メモリを管理する関数を以下に示します。

- uldb_free_namelist
- uldb_free_limit_list

uldb_free_namelist

uldb_free_namelist 関数は、namelist 構造体とそのコンポーネントをすべて削除します。

uldb_free_namelist の例を次に示します。

```
#include <include/uldb.h>
void                             # returns 0 if okay, -1 on error
   uldb_free_namelist (            #
      uldb_namelist_t *names);     # pointer to namelist to be freed
```

uldb_free_limit_list

uldb_free_limit_list 関数は、制限リスト構造体とそのコンポーネントをすべて削除します。

uldb_free_limit_list の例を次に示します。

```
#include <include/uldb.h>
void                                # returns 0 if okay, -1 on error
   uldb_free_limit_list (           #
      uldb_limit_list_t *limits);  # pointer to limit list to be freed
```

エラー・メッセージ

エラー・メッセージについては、uldb_get_limit_values(3c) や jlimit_startjob(3c) のマン・ページか47 ページの付録A「ジョブ制限に関するエラー・メッセージとマン・ページの概要」を参照してください。

Miser バッチ処理システム

Miser は、時間や領域の必要量がわかっているアプリケーションの決定性バッチ・スケジューリングを可能にするリソース管理機能で、システム・リソースの静的な分割が不要です。ジョブを指定すると、Miser は管理下の時間/領域のプール全体を検索し、そのジョブのリソース要件に最適な割り当てを求めます。

Miser には、再起動せずにほとんどのパラメータを修正できる、管理用の拡張インタフェースがあります。Miser は、独立したトラステッド・プロセス (高信頼プロセス) として動作します。カーネルからかユーザからかを問わず、Miser への通信はすべて一連の Miser コマンドを通じて行われます。Miser は、プロセスのスケジューリング、プロセスの状態変更、およびバッチ・システム設定の制御に対する要求を受信し、その要求の値とステータス情報を返します。

Miser の概要

Miser は、時間と領域からなるプールのセットを管理します。プールの時間コンポーネントは、Miser がどの程度先までジョブをスケジュールできるかを定義します。プールの領域コンポーネントは、ジョブをスケジュールできるリソースの集合です。領域コンポーネントは、時間によって変化します。

システム・プールは、Miser が利用できるリソース (CPU の数や物理メモリ) の集合を表します。ユーザ定義プールの集合は、ジョブをスケジュールできるリソースを表します。ユーザのプールが所有するリソースは、Miser が利用できるリソース合計を超えることはできません。Miser が管理するリソースがスケジュールされたジョブによって使用されていない場合は、Miser 以外のアプリケーションがそのリソースを利用できます。

各プールには、プール・リソースの定義、プールからリソースを割り当てるジョブの集合、ジョブのスケジューリングを制御するポリシーが関係付けられています。リソース・プール、スケジュールされたジョブ、ポリシーをまとめて**キュー** と呼びます。

キューを使用することで、バッチ・システムをきめ細かく管理することができます。キューに割り当てるリソースは、時間に応じて変えることができます。たとえば、日中は 5 個、夜間は 20 個の CPU を管理するようにキューを設定することができます。複数のキューを使用すると、バッチ・システムのユーザ間でリソースを分配することができます。たとえば、24 個の CPU があるシステム上で 2 つのキューを定義し、一方に 16 個の CPU、もう一方に 6 個の CPU を割り当てることができます (2 個の CPU は Miser の管理外に置くものとします)。キューへのアクセスをシステム上の特定ユーザまたはユーザのグループに制限することで、リソース分配をユーザやグループに強制することができます。

ポリシーは、アプリケーションのリソース要件を満たす時間/領域のブロックを検索する方法を定義します。Miser には、2 つのポリシー、“ default” と“ repack” があります。default は、ファースト・フィット・ポリシーで、いったんジョブがスケジュールされると、その開始と終了の時刻は変更できません。先行

するジョブがスケジュールよりも早く完了した場合でも、それ以後にスケジュールされているジョブの開始/終了の時刻には影響しません。一方 repack は、ファースト・フィット・ポリシーに加えて、スケジュールされたジョブの順序を保ちつつ、先行するジョブが早く終了した場合には、残りのジョブを前倒しするように再スケジュールを行います。

ユーザは、`miser_submit` コマンドを使用して、ジョブをキューに入れることができます。このコマンドは、ジョブを入れるキューと、キューに対して行われるリソース要求を指定します。それぞれの Miser ジョブは、IRIX プロセス・グループです。リソース要求は、時間と領域の組です。時間はシングル CPU で実行された場合の総 CPU 時間 (wall-clock 時間) です。領域は論理 CPU 数と要求される物理メモリです。要求は Miser に渡され、Miser は、キューに付与されたポリシーを使用して、キューのリソースに対してジョブをスケジュールします。Miser は、ジョブの開始時刻と終了時刻をユーザに返します。

ジョブの開始時刻にならないうちは、ジョブはバッチ状態にあります。バッチ状態にあるジョブの優先度は、実行中のどのプロセスよりも低くなります。システムに待ち状態のリソースがあれば、バッチ状態にあるジョブを実行できます。これを「日和見的に実行される (run opportunistically)」といいます。指定した実行時刻になると、ジョブの状態はバッチ・クリティカルに変わり、その時点でジョブの優先度はどの非リアルタイム・プロセスよりも高くなります。バッチ状態の実行で消費した時間は、要求およびスケジュールされた時間にはカウントされません。プロセスがバッチ・クリティカル状態にある間、要求した物理メモリと CPU の確保は保証されます。割り当て時間を超えるか、要求した物理メモリよりも多くのメモリを使用すると、プロセスは終了します。

default のポリシーを使用してスケジュールされた、static フラグの指定があるジョブは、そのセグメントの実行がスケジュールされた時刻にのみ実行されます。待ち状態のリソースが使用可能な場合でも、先行して実行されることはありません。repack ポリシーを使用してジョブをスケジュールする場合は、先行して実行することが可能です。

論理 CPU 数について

ジョブがスケジュールされると、Miser は、ジョブのためにいくつかの CPU と一定量のメモリを予約するように要求します。これらのリソースがジョブの開始に備えて予約されている間、Miser は特定の CPU と一定量の論理スワップ領域をジョブのために予約します。

ジョブへの CPU 割り当てに影響を及ぼす問題がいくつかあります。ジョブがバッチ・クリティカルになると、Miser は小範囲に密集しているノードのクラスタを探そうとします。そのようなクラスタが見つからない場合、そのジョブのスレッドは、利用可能な任意の空き CPU に割り当てられます。これらの CPU が、システムの離れた位置に分散している場合もあります。

対話型プロセスにおける CPU 予約の効果

Miser の利点の 1 つは、CPU 予約の方法です。Miser は、物理的な CPU 数ではなく論理 CPU 数に基づいて CPU の制御と予約を行います。これにより Miser は、CPU リソースを柔軟に制御することができます。

日和見的に実行される対話型とバッチ型のプロセスは、システム内で Miser のジョブ用にまだ予約されていない CPU を使用することができます。新しいジョブが入ると、Miser は、利用できる論理的なリソースの量に基づいて、ジョブをスケジュールします。その結果、CPU が Miser によって予約され、対話型プロセスは新たに予約された CPU 上でそれ以上実行できなくなります。しかし、リソースが Miser に使用されない場合、そのリソースは、他の任意のアプリケーションが自由に使用できます。Miser は、必要になったときにリソースを要求します。

Miser のメモリ管理について

CPU は必要に応じて Miser によって予約されますが、メモリは、必要になる前にあらかじめ予約しておく必要があります。

Miser は起動時にジョブのために予約できる CPU の数とメモリ量を知らされます。CPU の数は、論理数です。Miser のジョブがバッチ・クリティカルになると、CPU が割り当てられます。Miser のジョブで CPU が必要になるまでは (つまり、プロセスやスレッドが実行可能になるまでは)、システムの残りの部分がその CPU を利用できます。Miser のジョブのスレッドが実行を開始すると、現在の非 Miser スレッドは実行を一時停止され、Miser スレッドが現在実行されていない CPU 上で再開されます。

メモリ・リソースは、CPU リソースとまったく状況が異なります。Miser がジョブの予約に使用するメモリは、**論理スワップ・スペース**と呼ばれます。論理スワップ・スペースは、物理メモリ (カーネルによって占有される小さな領域) とすべてのスワップ・デバイスの総計として定義されます。

Miser の起動時にジョブに対してメモリを予約する必要があります。ただし、物理メモリを予約する必要はありません。非 Miser ジョブのメモリを移動するのに十分な、物理メモリとスワップがあることだけが確認されます。これは、必要なメモリに等しい論理スワップを予約することで行われます。

Miser に管理されるジョブだけが、Miser 用に予約された論理スワップ・スペースの割り当てを使用できます。Miser に使用されていない物理メモリは、他の任意のアプリケーションが自由に使用できます。Miser は、必要になったときに物理メモリを要求します。

Miser の管理がユーザに及ぼす影響

ユーザが Miser にジョブを指定すると、そのジョブに予約されたリソースが要求された時間の間、割り当てられます。ジョブは、システム・リソースを求めて競合する必要はありません。結果として、対話型ジョブとして実行する場合よりも、迅速かつ安定したジョブの実行が完了します。ただしデメリットもあります。Miser はリソースの領域共有であるため、要求したリソースが予約されるまでのスケジュール期間、ジョブは待ち状態になります。それよりも先に、非静的ジョブが空きを見て実行される場合があります。これは対話型の負荷と競合することになりますが、優先度は対話型の負荷よりも低くなります。

ユーザが対話的に作業している場合、そのユーザは、システム・リソースすべてを利用できるわけではなくなります。ユーザの対話型プロセスは、システム上で予約されていない CPU すべてを利用できます

が、メモリ割り当てに対して制限された量の論理スワップ・スペースしか持たないことになります。非 Miser ジョブで利用できる論理スワップ・スペースの量は、Miser の起動時に予約されなかった量になります。

Miser の設定

Miser で主として設定の対象となるのは、キューの集合です。Miser のキューは、Miser に割り当てられるリソースを定義します。

Miser の設定は、以下のように行います。

- Miser システム・キュー定義ファイルを設定する。Miser システムには、必ず Miser システム・キュー定義ファイルが必要です。このファイルのベクトル定義は、ほかのキューのベクトル定義で利用可能な最大リソースを指定します。
- Miser ユーザ・キュー定義ファイルを設定してキューを定義する。
- Miser 設定ファイルを設定して Miser システムの一部となるすべてのキューを列挙する。
- Miser コマンドライン・オプション・ファイルを設定して、Miser が管理できる最大の CPU 数とメモリを定義する。

Miser システム・キュー定義ファイルの設定

Miser システム・キュー定義ファイル (`/etc/miser_system.conf`) は、システム・プールによって管理されるリソースを定義します。このファイルは、プールの最大持続時間を定義します。他のすべてのキューは、システム・キュー以下でなければなりません。システム・キューは、ジョブが要求できるリソースの上限を指定します。Miser システム・キューが設定されている必要があります。

有効なトークンは、以下のとおりです。

POLICY *name*	システム・キューにはポリシーがないため、ポリシーは常に“ none”です。
QUANTUM *time*	量子時間 (quantum) のサイズ。**quantum** とは、任意の秒数を表す Miser の用語です。quantum は、時間/領域のプールを分割する方法を指定するために使用します。これは、システム・キュー定義ファイルとユーザ・キュー定義ファイルの両方に指定され、両方のファイルで同じでなければなりません。
NSEG *number*	リソース・セグメントの数。

SEGMENT	ベクトル定義の、新しいセグメントの先頭を定義します。各々の新しいセグメントは、*SEGMENT* トークンで始める必要があります。各セグメントには少なくとも CPU の数、メモリ、wall-clock 時間を含めなければなりません。
START *number*	セグメントが開始される量子時間。時刻の基点は、ローカル時間で 1970 年 1 月 1 日 (火) の 00:00 です。 Miser は、現在の日付になるまでキューを前方に繰り返すことにより、開始時刻と終了時刻を現在の時刻にマップします。たとえば、24 時間のキューは常に、現在の日付の午前零時から始まります。
END *number*	セグメントが終了する量子時間。
NCPUS *number*	CPU の数。
MEMORY *amount*	整数で指定されるメモリの量。k (キロバイト)、m (メガバイト)、g (ギガバイト) の単位を後に付けることができます。単位の指定がない場合は、デフォルトでバイト単位になります。

次に示すシステム・キュー定義ファイルは、20 秒の量子時間と 1 つの要素をベクトル定義に持つキューを定義します。開始時刻と終了時刻は、秒単位ではなく量子時間単位で指定されています。

このセグメントは、1 個の CPU と 5 メガバイトのメモリがある、00:00 に開始して 00:20 に終了するリソース単位を定義しています。

```
POLICY none # System queue has no policy
QUANTUM 20 # Default quantum set to 20 seconds
NSEG 1

SEGMENT
START 0
END 60# Number of quanta (20min*60sec) / 20
NCPUS 1
MEMORY 5m
```

Miser ユーザ・キュー定義ファイルの設定

Miser ユーザ・キュー定義ファイル (`/etc/miser_default.conf`) は、CPU、物理メモリ、ポリシー名、キューのリソース・プールを定義します。このファイルは、キューのポリシーを指定するヘッダ、リソース・セグメントの数、キューに使用される量子時間で構成されます。

キューへのアクセスは、キュー定義ファイルのファイル・アクセス権限によって管理されます。読込み権限があると、ユーザは `miser_qinfo` コマンドを使用して、キューの内容を調べることができます。実行権限があると、ユーザは `miser_submit` コマンドを使用して、キューにジョブをスケジュールできます。

書込み権限があると、ユーザは miser_move コマンドと miser_reset を使用して、キューのリソースを変更することができます。

デフォルトのユーザ・キュー定義ファイルは、ほかのユーザ・キュー定義ファイルのテンプレートとして使用できます。各々の Miser キューには、独立したキュー定義ファイルがあります。このファイル名は、全般的な Miser 設定ファイル (/etc/miser.conf) で指定します。

ユーザは、ユーザ・キューによって管理されるリソースに対してスケジュールを行います。システム・キューに対して行うのではありません。ユーザ・キューによって指定される持続時間が、システム・キューによって指定されるものよりも短い場合、ユーザ・キューは何度も繰り返されます (たとえば、システム・キューが 1 週間を指定し、ユーザ・キューが 24 時間を指定した場合など)。システム・キューがユーザ・キューで割り切れない (たとえば、システム・キューが 6 で、ユーザ・キューが 5) 場合は、余りを切り捨てた整数回だけ繰り返します。

有効なトークンは、以下のとおりです。

POLICY *name*
: キューに入れられるアプリケーションをスケジュールするために使用するポリシーの名前。有効なポリシーは、“ default” と ” repack” の 2 つです。default はファースト・フィット・ポリシーです。これは、いったんジョブがスケジュールされると、その開始と終了の時刻が一定のままになる指定です。repack はスケジュールされたジョブの順序を維持しつつ、先行するジョブが早く終了した場合には、残りのジョブを前倒しするように再スケジュールを試みます。最初にジョブをスケジュールするときは、どちらのポリシーもファースト・フィット方式を使用する点に注意してください。

QUANTUM *time*
: 量子時間 (quantum) のサイズ。**quantum** は、任意の秒数を表す Miser の用語です。quantum は、時間/領域のプールを分割する方法を指定するために使用します。これは、システム・キュー定義ファイルとユーザ・キュー定義ファイルの両方に指定され、両方のファイルで同じでなければなりません。

NSEG *number*
: リソース・セグメントの数。

SEGMENT
: ベクトル定義の、新しいセグメントの先頭を定義します。各々の新しいセグメントは、*SEGMENT* トークンで始める必要があります。各セグメントには少なくとも CPU の数、メモリ、wall-clock 時間を含めなければなりません。

START *number*
: セグメントが開始される量子時間。時刻の基点は、ローカル時間で 1970 年 1 月 1 日 (火) の 00:00 です。

  Miser は、現在の日付になるまでキューを前方に繰り返すことにより、開始時刻と終了時刻を現在の時刻にマップします。たとえば、24 時間のキューは常に、現在の日付の午前零時から始まります。

END *number*
: セグメントが終了する量子時間。

NCPUS *number*	CPU の数。
MEMORY *amount*	整数で指定するメモリの量。k (キロバイト)、m (メガバイト)、g (ギガバイト) の単位を後に付けることができます。単位の指定がない場合は、デフォルトでバイト単位になります。

次に示すシステム・キュー定義ファイルでは、“ default” という名前のポリシーを使用してキューを定義しています。このキューは、20 秒の量子時間と 3 つの要素をベクトル定義に持っています。各セグメントの開始時刻と終了時刻は、秒単位ではなく量子時間単位で指定されています。

- 最初のセグメントは、50 個の CPU と 100 メガバイトのメモリがある、00:00 に開始して 00:50 に終了するリソース単位を定義しています。
- 2 番目のセグメントは、50 個の CPU と 100 メガバイトのメモリがある、00:51.67 に開始して 01:00 に終了するリソース単位を定義しています。
- 3 番目のセグメントは、50 個の CPU と 100 メガバイトのメモリがある、01:02.00 に開始して 01:03.33 に終了するリソース単位を定義しています。

```
POLICY default
QUANTUM 20
NSEG 3

SEGMENT
START 0
END 150 (50min*60sec) / 20
NCPUS 50
MEMORY 100m

SEGMENT
START 155 ((51min*60sec)+67) / 20
END 185 (1h*60min*60sec) / 20
NCPUS 50
MEMORY 100m

SEGMENT
START 186 ((1h*60min*60sec)+(2min*60sec)) / 20
END 190 ((1h*60min*60sec)+(3min*60sec)+33sec) / 20
NCPUS 50
MEMORY 100m
```

Miser 設定ファイルの設定

Miser 設定ファイル (/etc/miser.conf) には、Miser キューの名前と、各キューに対するキュー定義ファイルのパス名を記載します。このファイルには、すべてのキュー名とそのキュー定義ファイルが列挙されます。

すべての Miser 設定ファイルには、キューの 1 つとして、システム・プールのリソースを定義する Miser システム・キューを含める必要があります。Miser システム・キューは、“ system” という名前で指定します。

有効なトークンは、以下のとおりです。

QUEUE *queue_name queue_definition_file_path*

> *queue_name* には、Miser へのインタフェースで使用するキューの名前を指定します。キュー名は、1 文字以上、8 文字以下でなければなりません。キュー名“ system” は、Miser のシステム・キューを指定するために使用します。

Miser 設定ファイルのサンプルを以下に示します。

```
# Miser config file
QUEUE system /hosts/foobar/usr/local/data/system.conf
QUEUE user /hosts/foobar/usr/local/data/usr.conf
```

Miser コマンドライン・オプション・ファイルの設定

Miser コマンドライン・オプション・ファイル (/etc/config/miser.options) は、Miser が管理できる最大の CPU 数とメモリを定義します。

-c フラグは、Miser が使用できる CPU の最大数を定義します。この値は、システム・キューのリソース・セグメントが予約できる CPU の最大数です。

-m フラグは、Miser が使用できる最大のメモリを定義します。この値は、システム・キューのリソース・セグメントが予約できる最大メモリです。Miser 用に予約されるメモリは、物理メモリから割り当てられます。Miser が使用するメモリの量は、物理メモリの総量からカーネルが使用するのに十分なメモリを除いた量よりも少なくなければなりません。また、Miser のメモリがすべて使用中の場合に、Miser の管理下にない既存プロセスを掃き出すのに十分なスワップ・スペースが確保されるように、システムには少なくとも Miser が設定する量のスワップ・スペースが必要です。

次に示す例では、コマンドライン・オプション・ファイルで -c と -m の値をそれぞれ、*1* と *5* メガバイトに設定しています。

```
-f/etc/miser.conf -v -d -c 1 -m 5m
```

-v フラグは、冗長モードを指定します。これにより詳細情報が出力されます。

-d フラグは、デバッグ・モードを指定します。このモードを指定すると、アプリケーションは tty の制御を放棄しません (つまり、デーモンにはなりません)。このモードは、Miser が正しく起動しない原因を調べる際に、-v フラグと組み合わせて使用すると便利です。

メモ: -C フラグを使用すると、Miser デーモンを終了 (kill) した後、それが再起動される前に、Miser が予約したすべてのリソースを解放することができます。詳細については、miser(1) のマン・ページを参照してください。

設定の指針

Miser の設定は、サイトによって異なります。以下の指針を参考にしてください。

- 対話/バッチプロセスの利用率について、システムのバランスをとる必要があります。1 つの考え方は、Miser の制御下にないプロセッサを少なくとも 1 つか 2 つ、常に残しておくということです。これらのプロセッサは、Miser が管理する CPU がすべて予約されている場合に、システムの対話型の部分として機能します。対話型の処理では、CPU に必要な負荷の平均は、一般に 2 よりも小さくなります。空き CPU のオプション数の調整にあたっては、この点に留意してください。
- 論理スワップの空き容量は、空き CPU の数とのバランスをとる必要があります。システムに *N* 個の CPU がある場合、これら *N* 個の CPU で実行されるプロセスの使用量に相当するメモリも必要になります。また、多くのシステム管理者は、スワップ・スペースを使用してこのメモリをバックアップします。空き CPU を独立したシステムととらえて、それに応じたメモリとスワップ・スペースを用意すれば、対話的な処理は問題なく機能します。Miser に予約されない空きメモリは、論理スワップ・スペース (物理メモリとスワップ・デバイスの組み合わせ) であることを忘れないでください。
- 仮想スワップを使用する場合は、注意が必要です。Miser アプリケーションが実行されていない場合、タイムシェア・プロセスがすべての物理メモリを消費する可能性があります。Miser が実行されると、Miser は物理メモリの返還を要求し、タイムシェア・プロセスをスワップ・アウトします。システムが仮想スワップを使用している場合は、プロセスを移動する物理スワップがない可能性があり、この時点でタイムシェア・プロセスは終了することになります。

Miser の設定例

この節で使用する例では、ユーザ・プログラム用に 12 個の CPU と 160 MB のメモリが使用可能であるものとします。

例 1:

この例では、システムが、1 つのキューを使用した、1 日 24 時間のバッチ・スケジューリング専用になっています。

最初のステップでは、システム・キューを定義します。システム・キューに指定する長さを決定する必要があります。システム・キューの長さは、システムによって管理されるジョブの最大持続時間を定義します。このシステムの場合、1 つのジョブに対して最大 48 時間の持続時間という判断から、システム・ベクトルが 48 時間の持続時間となるように定義しています。

```
# The system queue /usr/local/miser/system.conf
POLICY none # System queue has no policy
QUANTUM 20 # Default quantum set to 20 seconds
NSEG 1

SEGMENT
NCPUS 12
MEMORY 160m
START 0
END 8640 # Number of quanta (48h*60 min*60 sec) / 20
```

次のステップでは、ユーザ・キューを定義します。

```
# The user queue /usr/local/miser/physics.conf
POLICY default # First fit, once scheduled maintains start/end time
QUANTUM 20 # Default quantum set to 20 seconds
NSEG 1

SEGMENT
NCPUS 12
MEMORY 160m
START 0
END 8640 # Number of quanta (48h*60 min*60 sec) / 20
```

最後のステップは、Miser 設定ファイルの定義です。

```
# Miser config file
QUEUE system /usr/local/miser/system.conf
QUEUE physics /usr/local/miser/physics.conf
```

例 2:

次の例では、システムが、1 日 24 時間のバッチ・スケジューリング専用になっており、2 つのユーザ・グループ chemistry と physics に分かれています。システムは、physics に 75%、chemistry に 25% の比率で、それぞれに分配するものとします。

システム・キューは、例 1 で示したものと同一です。

physics ユーザ・キューは、次のようになります。

```
# The physics queue /usr/local/miser/physics
POLICY default # System queue has no policy
QUANTUM 20 # Default quantum set to 20 seconds
NSEG 1

SEGMENT
NCPUS 8
MEMORY 120m
START 0
END 8640 # Number of quanta (48h*60min*60sec) / 20
```

次に、chemistry キューを定義します。

```
# The chemistry queue /usr/local/miser/chemistry.conf
POLICY default # System queue has no policy
QUANTUM 20 # Default quantum set to 20 seconds
NSEG 1

SEGMENT
NCPUS 4
MEMORY 40m
START 0
END 8640 # Number of quanta (48h*60min*60sec) / 20
```

各キューへのアクセスを制限するため、ユーザ・グループ physics と chemistry を作成します。次に、physics のキュー定義ファイルには、グループ physics にのみ実行許可を設定します。chemistry キューに対しても同様の設定を行います。

physics と chemistry のキューを定義したので、Miser 設定ファイルの定義を行います。

```
# Miser configuration file
QUEUE system /usr/local/miser/system.conf
QUEUE physics /usr/local/miser/physics.conf
QUEUE chem /usr/local/miser/chemistry.conf
```

例 3:

この例では、システムが、昼間はタイムシェアリング専用、夜間はバッチ専用となります。夜間は午後 8:00 から 午前 4:00 まで、昼間は午前 4:00 から午後 8:00 までです。

最初に、システム・キューを定義します。

```
# The system queue /hosts/foobar/usr/local/data/system.conf
POLICY none # System queue has no policy
```

```
QUANTUM 20 # Default quantum set to 20 seconds
NSEG 2

SEGMENT
NCPUS 12
MEMORY 160m
START 0
END 720 # (4h*60min*60sec) / 20

SEGMENT
NCPUS 12
MEMORY 160m
START 3600 # (8pm is 20 hours from UTC, so 20h*60min*60sec) / 20
END 4320
```

次に、バッチ・キューを定義します。

```
# User queue
POLICY repack # Repacks jobs (FIFO) if a job finishes early
QUANTUM 20 # Default quantum set to 20 seconds
NSEG 2

SEGMENT
NCPUS 12
MEMORY 160m
START 0
END 720 # (4h*60min*60sec) / 20

SEGMENT
NCPUS 12
MEMORY 160m
START 3600 # (8pm is 20 hours from 0, so 20h*60min*60sec) / 20
END 4320
```

最後のステップは、Miser 設定ファイルの定義です。

```
# Miser config file
QUEUE system /usr/local/miser/system.conf
QUEUE user /usr/local/miser/usr.conf
```

Miser の起動と停止

Miser 設定ファイルに必要な修正を加えた後、chkconfig(1) コマンドを使用してブート時の起動対象に選択し、システムを再起動します。あるいは、コマンド /etc/init.d/miser start を使用して、**root** が直接 Miser を起動することもできます。再起動せずに直接 Miser を手動で起動する場合は、最初に chkconfig コマンドを発行する必要があり、これを怠ると Miser は起動しません。

Miser を手動で起動するには、次のコマンド・シーケンスを使用します。

```
# chkconfig miser on
# /etc/init.d/miser start
```

root は、次のコマンド・シーケンスを使用して、いつでも Miser を停止できます。

```
# /etc/init.d/miser stop
# /etc/init.d/miser cleanup
```

実行中の Miser ジョブは停止されません。現在使用中のリソースは、ジョブが終了するまでは返還要求できません。停止した後で、再度 Miser を起動する場合は、miser cleanup コマンドを実行する必要はありません。

メモ: Miser の -C フラグを使用すると、Miser デーモンを終了 (kill) した後、それが再起動される前に、Miser が予約したすべてのリソースを解放することができます。

Miser ジョブの指定

Miser によって管理されるようにジョブを指定するコマンドは、次のようになります。

```
miser_submit -q queue -o c=cpus, m=memory, t=time [,static] command
miser_submit -q queue -f file command
```

-q *queue*　　アプリケーションをスケジュールするキューの名前を指定する。

`-o c=`*cpus*`,m=`*memory*`,t=`*time*`[,static]`	リソースのブロックを指定する。*cpus* は、スケジュールの対象となるキューで利用できるCPUの最大数以下の整数です。メモリ量 *memory* は、**k** (キロバイト)、**m** (メガバイト)、**g** (ギガバイト) の単位を後に付けた整数で指定されます。単位を付けない場合は、デフォルトでバイト単位となります。*time* には、**h** (時)、**m** (分)、または **s** (秒) のいずれかを整数の後に付けたものか、*hh*:*mm*:*ss* フォーマットの文字列で指定します。 default ポリシーを使用してスケジュールされた、`static` フラグの指定があるジョブは、そのセグメントが実行するようにスケジュールされた時刻にのみ実行されます。待ち状態のリソースがジョブを利用できる場合でも、先行して実行されることはありません。repack ポリシーを使用してジョブをスケジュールする場合は、先行して実行することが可能です。
`-f` *file*	リソース・セグメントのリストを指定するファイル。このフラグを使用すると、ジョブのスケジューリング・パラメータを詳しく制御できます。
command	スクリプトやプログラムの名前を指定します。

詳細については、`miser_submit`(4) や `miser_submit`(1) のマン・ページを参照してください。

ジョブのスケジュール/説明に関する Miser への問合せ

指定したジョブのスケジュール/説明について、Miser に問合せを行うコマンドは、次のようになります。

```
miser_jinfo -j bid [-d]
```

bid は、Miser ジョブの ID であり、ジョブのプロセス・グループ ID です。`-d` フラグを指定すると、ジョブの所有者とコマンドを含む、ジョブの説明が表示されます。

システムが高負荷下にあるときは、Miser のスワップ処理にある程度の時間がかる場合があることに注意してください。したがって、Miser のジョブは、指定した後、即座に処理されない場合があります。

詳細については、`miser_jinfo`(1) のマン・ページを参照してください。

キューに関する Miser への問合せ

Miser のキューに関する情報、キューのリソース・ステータス、キューにスケジュールされたジョブの一覧を Miser に問い合せるコマンドは、次のようになります。

```
miser_qinfo -Q|-q queue [-j]|-a
```

-Q フラグを指定すると、現在設定されている Miser のキュー名の一覧を返します。-q フラグを指定すると、指定したキュー名に対応する空きリソースを返します。-j フラグを指定すると、そのキューに現在スケジュールされているジョブの一覧を返します。-a フラグを指定すると、設定済みのすべての Miser キューにある、すべてのスケジュール済みのジョブをジョブ ID の順序で並べた一覧を返します。ジョブの簡単な説明も出力されます。

詳細については、miser_qinfo(1) のマン・ページを参照してください。

リソースのブロックの移動

あるキューから別のキューにリソースのブロックを移動するコマンドは、次のようになります。

```
miser_move -s srcq -d dstq -f file
miser_move -s srcq -d dstq -o s=start, e=end, c=CPUs, m=memory
```

このコマンドは、始点の時間 *start* から始めて終点の時間 *end* まで、移動元キュー *srcq* のベクトルから領域の組を削除し、それを移動先キュー *dstq* のベクトルに追加します。追加または削除されるリソースは、ベクトル定義は変更しません。したがって、それらは一時的なものです。このコマンドは、各リソース移動の開始時刻と終了時刻と、転送したリソースの量を返します。

-s と -d のフラグには、任意の有効な Mister キューを指定できます。-f フラグには、リソース・ブロックの指定が含まれます。-o フラグには、移動するリソースのブロックを指定します。開始と終了の時刻は、現在時刻からの相対値で指定します。*CPUs* には、キューに関連付けられた空き CPU の最大数までの整数が指定できます。メモリは、**k** (キロバイト)、**m** (メガバイト)、**g** (ギガバイト) の識別子が付いた整数です。

メモ:リソースの移動は一時的なものです。Miser が終了 (kill) させられるかクラッシュした場合、移動されたリソースは失われ、Miser は再起動できなくなります。

詳細については、miser_move(4) や miser_move(1) のマン・ページを参照してください。

Miser のリセット

新しい設定ファイルを使用して Miser をリセットするためのコマンドは、次のようになります。

```
miser_reset -f file
```

このコマンドは、新しい設定ファイル (-f *file* で指定される) を、実行中の Miser のバージョンに対して強制的に適用します。すべてのスケジュール済みのジョブが、新しい設定に対してうまくスケジュールできる場合にのみ、新しい設定は成功します。

詳細については、miser_reset(1) のマン・ページを参照してください。

Miser ジョブの終了

miser_kill コマンドを使用すると、Miser に対して指定したジョブを終了させることができます。このコマンドは、プロセスを終了させると同時に、Miser デーモンと通信して、指定済みのプロセスが現在使用しているすべてのリソースを解放します。詳細については、miser_kill(1) のマン・ページを参照してください。

Miser とバッチ管理システム

この節では、Miser のジョブと、バッチ管理システムのバッチ・ジョブとの違いについて説明します。バッチ管理システムの例としては、Network Queuing Environment (NQE) や Load Share Facility (LSF) などがあります。

Miser と NQE などのバッチ管理システムは、互いに異なる重要な特質に欠けています。Miser の場合、Miser セッションを保護、管理する機能がありません。これに対してバッチ管理システムの場合は、リソースを確保する能力が欠けています。しかし、バッチ管理システムが Miser のスケジューラに対応すれば、これら 2 つのシステムを一緒に使用することで、より能力の高いソリューションを提供することができます。

ユーザのサイトで、バッチ管理システムによって提供されるジョブの管理や保護が必要ないのであれば、Miser が単独で十分なバッチ・システムとなります。しかし、製品レベルの品質が求められるほとんどの環境では、NQE や LSF などのバッチ・システムによって提供されるサポートや保護が必要になります。こうしたサイトでは、Miser のスケジューラと一緒にバッチ管理システムを実行する必要があります。

cpuset の定義と管理

cpuset コマンドは cpuset と呼ばれる、CPU の集合を定義し、管理することができます。cpuset は、CPU の名前付きの集合で、制限付きまたは開放されたものとして定義できます。cpuset コマンドを使用すると、cpuset の作成や破棄、既存の cpuset に関する情報の取り出し、およびプロセスやその子プロセスを cpuset に結び付けることができます。

メモ : cpuset コマンドでは、Miser バッチ処理システムを使用する必要はありません。

制限付きの cpuset では、その cpuset のメンバーになっているプロセスしか、該当する CPU 集合の上で実行できません。開放された cpuset の場合は、任意のプロセスがその CPU 上で実行できますが、cpuset のメンバーであるプロセスはその cpuset に所属する CPU 上でのみ実行できます。

cpuset は、cpuset 設定ファイルと名前によって定義されます。ファイル・フォーマットの定義については、cpuset(4) のマン・ページを参照してください。cpuset 設定ファイルは、その cpuset のメンバーとなる CPU を列挙するために使用します。このファイルには、cpuset を定義するのに必要な追加の引数も含まれます。cpuset 名の長さは、3 文字以上、8 文字以下です。2 文字以下の名前は、予約されています。

設定ファイルの権限により、cpuset へのアクセスが定義されます。権限をチェックする必要がある場合、ファイルの現在のパーミッションが使用されます。したがって、特定の cpuset へのアクセスは、それを破棄して作り直さなくても、単にアクセス権限を変更するだけで変更できます。読込みアクセスがあれば、ユーザは cpuset に関する情報を取得でき、実行権限があれば、cpuset にプロセスを結びつけることができます。

次の例は、3 個の CPU を含む排他的な cpuset を記述するサンプル構成ファイルです。

```
# cpuset configuration file
EXCLUSIVE
MEMORY_LOCAL
MEMORY_EXCLUSIVE

CPU 1
CPU 5
CPU 10
```

この指定により、3 個の CPU を含む cpuset が作成されます。EXCLUSIVE フラグを設定すると、上記の CPU は、この cpuset 専用に割り当てられたスレッドの実行に制限されます。MEMORY_LOCAL フラグを設定すると、この cpuset 上で実行されるジョブは、cpuset の CPU を含むノードのメモリを使用します。MEMORY_EXCLUSIVE フラグを設定すると、他の cpuset やグローバルな cpuset で実行されるジョブは、これらのノードのメモリを使用しません。

メモ: Miser と cpuset の両方があるシステムでは、Miser キューが使用する CPU と、cpuset に割り当てられた CPU との間で競合が発生する場合があります。Miser は、EXCLUSIVE フラグ設定で構成された cpuset に所属する CPU にはアクセスしません。

コマンドは改行で区切られ、コメント区切りの # に続く文字は無視されます。大文字と小文字は区別され、トークンは空白 (無視される) によって切り出されます。

有効なトークンは、以下のとおりです。

有効なトークン	説明
EXCLUSIVE	制限付きの cpuset を定義する。ファイル内の任意の場所に記述できます。同じ行にあるそれ以外の部分は無視されます。
MEMORY_LOCAL	cpuset に割り当てられたスレッドは、その cpuset 内部のノードだけからメモリを割り当てるようにする。cpuset の外部からのメモリ割り当ては、cpuset 内で空きメモリが利用できない場合にのみ発生します。cpuset の外部で実行しているスレッドへのメモリ割り当てには制限は加えられません。

MEMORY_EXCLUSIVE	cpuset の外部でメモリが利用できない場合を除き、cpuset に割り当てられないスレッドは、その cpuset 内のメモリを使用しない。 cpuset が作成され、その cpuset のノードですでに実行中のスレッドによってメモリが占有されると、明示的にこのメモリを移動するための試みは何も行われません。ページ移動が可能な場合、非ローカルになっているページに多くの参照が行われていることをシステムが検出すると、そのページが移動されます。
MEMORY_KERNEL_AVOID	カーネルは、この cpuset を含んでいるノードからのメモリ割り当てを避ける。カーネルのメモリ要求が、この cpuset の外部では満たされない場合、このオプションは無視され、cpuset 内部でのメモリ割り当てが発生します。(この回避動作は現在、保護されたノードからバッファ・キャッシュを隔離するだけです。)
CPU	この cpuset の一部となる CPU を指定します。

cpuset コマンドの引数やその詳細については、cpuset(4) や cpuset(1) のマン・ページを参照してください。

ジョブ制限に関するエラー・メッセージとマン・ページの概要

リソース管理に関する製品や IRIX の機能を次に示します。

エラー・メッセージ

ジョブ制限に関する以下のエラー・メッセージが返されます。

EBUSY	要求されたジョブ ID の値が使用中。
EINVAL	無効なパラメータ。
ENOATTR	ドメイン名またはドメイン名リストが指定されていない。
ENOEXIST	`jlimits` ファイルが存在しない。
ENOJOB	指定されたジョブ ID の付いたジョブが見つからない。
ENOMEM	十分なメモリが利用できない。
ENOPKG	ジョブ制限ソフトウェアがインストールされていない。

マン・ページ

`man` コマンドを使用すると、すべてのリソース管理コマンドについてオンライン・ヘルプを参照できます。マン・ページをオンラインで表示するには、`man` コマンド名と入力します。

一般ユーザ用マン・ページ

ジョブ制限ソフトウェアでは、以下の一般ユーザ用マン・ページが用意されています。

一般ユーザ用マン・ページ	説明
`jlimit`(1)	リソース制限の表示と設定を行う
`jstat`(1)	ジョブのステータス情報を表示する
`showlimits`(1)	ユーザ制限データベースから制限情報を表示する

管理者用マン・ページ

ジョブ制限ソフトウェアでは、以下の管理者用マン・ページが用意されています。

管理者用マン・ページ	説明
genlimits(1M)	ユーザ制限データベースを作成する

アプリケーション・インタフェースに関するマン・ページ

ジョブ制限ソフトウェアを使用するアプリケーションの開発者向けに、以下のオンライン・マン・ページが用意されています。

アプリケーション・インタフェースに関するマン・ページ	説明
getjid(2)	ジョブ ID を取得する
getjlimit(2)	ジョブの最大システム・リソース消費量を制御する
getjusage(2)	ジョブの使用状況に関する情報を取得する
killjob(2)	指定したジョブのすべてのプロセスを終了させる
makenewjob(2)	新しいジョブ・コンテナを作成する
jlimit_startjob(3c)	新しいジョブを作成する
uldb_get_limit_values(3c)	ユーザ制限データベース (ULDB) と対話して、ドメインやユーザの制限値の取り出しや設定を行う機能の集合

索引

ふ