lkstlogtools Version 1.0.1

使用説明書

Rev. 1

履歴

version	日付	作成者	変更内容
1.0.0 rev.0	2005/2/20	平松、杉田	初版
1.0.1 rev.0	2005/2/22	平松	(開発版にあわせて更新) ・bioqueue アナライザを追加
1.0.1 rev.1	2005/03/15	平松	・lkstla の-l のオプションの記述を追加

―目次―

1. LKST 概要

1.1 背景・問題点

サーバ分野において、Linux を適用したシステムが普及・拡大している。数年前までは、Web サーバやメールサーバ、ネームサーバといったネットワークのフロントエンドサーバへの適用が中心であったが、最近では、アプリケーションサーバ、DB サーバから構成されるエンタープライズシステムへの適用ニーズも出てきている。

しかし、エンタープライズシステムでは迅速な障害対応が求められるが、Linux にはダンプやトレースといった障害解析のための標準的なツールが無く、障害発生時は、各社固有のノウハウで対応している。Linux 内の挙動情報を取得するツールは、我々が開発しコミュニティに提供している LKST(Linux Kernel State Tracer)など、いくつか提供されているが、性能障害に関しては適切なツールがない。性能障害発生時に十分なデータが得られないため、解決に時間がかかり、原因の特定に至らないケースも多々ある。すなわち以下のような問題点が在る。

- Linux には、カーネルの性能を評価する標準的な手段がサポートされていない
- 取得したい情報を、どのツール・情報で得られるかの指標がない
- 不足している情報を補う手段がない
- 性能評価技術は各社、各人のスキルであり、評価ノウハウが共有できていない

この問題を解決するため、以下の環境を提供する必要がある。

- 容易に性能情報を取得できる環境
- 新規情報の取得・解析の手段を容易に追加作成できる環境
- 性能評価手法を蓄積し共有できる環境

1.2 目的・方針

ミッション・クリティカル Linux を実現するために、OSS 適用システムの障害解析ツールとして、LKST によるカーネル性能評価ツールを開発する。既存の LKST の仕組みを活用し、性能評価機能部分を拡張する方法で実現する。また、必要な解析ツールの開発を行う。具体的には以下を実施する。

（1）機能開発
- LKST を利用したカーネル性能評価機能
- 性能情報を解析するツール

（2）ドキュメントによるノウハウの共有
- カーネル開発者、システム構築者にとって有益なカーネル性能評価のポイント
- LKST のポーティング、拡張開発方法

1.3　プロジェクトのスコープ

1.3.1　機能要件

機能として必要な事項・条件は以下の通り。

(1) アプリケーション処理中のカーネル処理部分の性能評価機能
  (a) システムコールの開始から処理終了までの時間
  (b) ＩＯ要求から終了までの時間
  (c) ユーザプロセスの生成から終了までの時間、など
  (d) 上記処理中のカーネル処理時間の内訳
    (i) メモリ確保に要する時間、実行回数
    (ii) ロックの取得に要する時間、ロックしている時間、実行回数
    (iii) プロセスの状態変化と各状態でいる時間

1.3.2　稼動要件

（１）ハードウェア条件：

・　IA-32 をベースとし、2GB 以上のメモリを搭載した PC

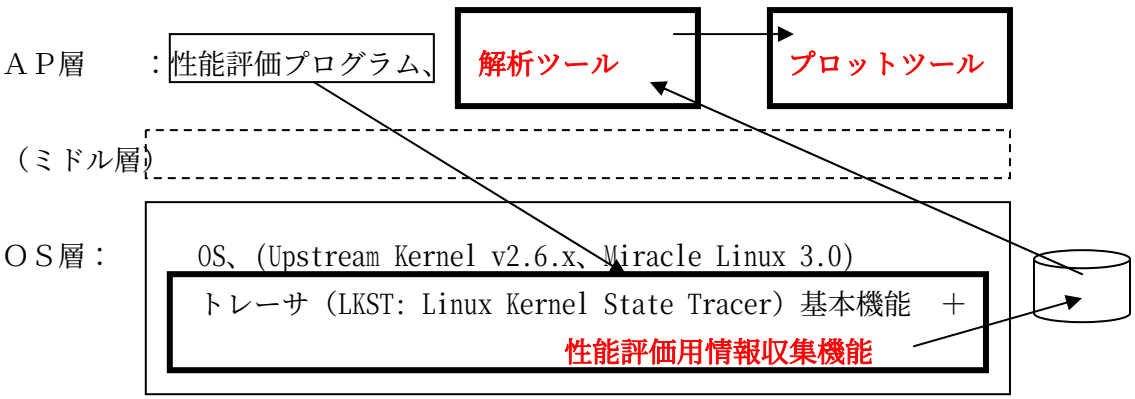
※メモリは 64MB 以上あれば動作可能。

（２）評価 Linux カーネルバージョン：

・　Upstream Kernel v2.6.x
　　- LKST version 2.2.1
・　Miracle Linux 3.0 (Asianux 1.0): Kernel v2.4.21
　　- LKST version 2.2.1 on Asianux

1.4　ソフトウェア構成概要

1.4.1　LKST 性能評価機能の構成



1.4.2 性能評価用情報収集機能：LKST 拡張機能

従来の LKST に対し、36 個のイベントを追加・修正して、システムの性能評価情を収集できるようにした。これらのイベントの詳細については「付録C：計測フックポイント・イベント一覧」を参照。

1.4.3 解析ツール（lkstla）

lkstla は、LKST のコマンド lkstbuf と lkstlogd が出力したバイナリログデータを解析し、あるイベントの分布や平均値などを出力する。

1.4.4 プロットツール

(1) 時系列グラフプロットツール:lkst_plot_log
lkstlaで解析した時系列ログから､gnuplotでグラフとしてプロットしたものをpdf化し、表示する。

(2) 統計結果グラフプロットツール:lkst_plot_stat
lkstla で解析した統計結果から、gnuplot でグラフとしてプロットしたものを pdf 化し、表示する。

(3) 対数分布ヒストグラムプロットツール:lkst_plot_dist
lkstla で解析した対数分布結果を、gnuplot でヒストグラムとしてプロットしたものを pdf 化し、表示する。

(4) システム情報グラフプロットツール：lkst_plot_sysinfo
lkst_fmt_sysinfo で解析したシステム情報の時系列ログから､gnuplot でグラフとしてプロットしたものを pdf 化し、表示する。

1.4.5 補助ツール

(1) システム情報解析ツール：lkst_fmt_sysinfo
LKST の拡張機能 lksteh_sysinfo.ko で記録した情報を解析し、時系列で出力する。

(2) ファイル分割ツール:lkstlogdiv
ログファイルの中で、時系列順が逆になる部分を境目にファイルを n 分割する。

(3) マスクセット作成ツール：lkst_make_mask
lkstla で解析するためのデータを記録するマスクセットを設定する。

1.5 既存ツール（Oprofile 等）との住み分け

1.5.1 代表的な既存ツールの概要

(1) OProfile（RedHat の Web ページからの抜粋）

http://squidward.mit.edu/rhel-doc/RH-DOCS/rhel-sag-ja-3/ch-oprofile.html

OProfile は、システム・パフォーマンス監視ツールである。プロセッサのパフォーマンス監視ハードウェアを使用してメモリ参照時、L2 キャッシュ要求数、ハードウェア割り込み回数など、システム上のカーネルや実行可能ファイルの情報を読み出す。

プロセッサの多くには専用のパフォーマンス監視ハードウェアが内蔵されている。このハードによって、特定のイベント(例えば要求データがキャッシュにない場合など)の発生を検出すること可能である。このハードウェアは通常、イベント発生毎に増分される 1 つ以上のカウンタ によって構成される。カウンタ値が"変わる"時に割り込みを行なえるため、パフォーマンス監視によって生成される詳細情報量(とオーバーヘッド)を制御することが可能である。

OProfile はこのハードウェア(パフォーマンス監視ハードウェアがない場合は、タイマベースの代用品)を使用して、カウンタが割り込みをカウントするたびパフォーマンス関連データのサンプルを収集して報告する。サンプルは周期的にディスクに書き込まれる。サンプルのデータは後にシステムレベル、アプリケーションレベルのパフォーマンスレポート生成に使用される。

(2) sar

sar コマンドは、システムに関する統計データを収集する。 sar コマンドは、システム・パフォーマンスに関する一部の有用なデータを収集できるが、サンプリングの頻度を高くすると、システム負荷が増加して、パフォーマンス上の問題を悪化させる可能性がある。

sar コマンドには多数のオプションがあり、キューイング、ページング、TTY、およびその他多数の統計情報を提供する。例えば sar コマンドは、システム全体の CPU 統計情報か、または各 CPU の統計情報のいずれかを報告する（SMP システムで有効)。

(3) vmstat

vmstat コマンドは、実行待ち行列および待機待ち行列、メモリ、ページング、ディスク、割り込み、システムコール、コンテキスト切り替え、および CPU アクティビティにおけるカーネル・スレッドに関する統計情報を報告する。報告される CPU アクティビティは、ユーザ・モード、システム・モード、アイドル時間、およびディスク入出力の待ちの分類パーセンテージである。

(4) iostat

iostat コマンドは、システムのディスク入出力の性能に関する統計情報を報告する。報告内

容は、物理ディスクがアクティブだった時間のパーセンテージ 、ドライブに転送されたデータの量 (KB/ 秒)、1 秒当たりの転送の数、測定間隔中に物理ボリュームから読み書きしたデータの合計 (KB 単位)である。

vmstat の CPU アクティビティと同じ情報も報告可能。

(5) /proc の下の情報

Linux ではシステムの稼動情報を/proc ファイルシステムで得ることができる。

/proc は擬似ファイルシステムであり、ディスク上にあるファイルシステムではないが、通常のファイルと同じようにアクセスが可能である。

/proc の下には、CPU 情報、メモリ情報、IO ポート割り当て情報、IRQ の割り当て状況、プロセスの状況などを得ることができる。従って、それらをタイムサンプリングで取得するスクリプトなどを常に動作させておく必要がある。

1.5.2 既存ツールとの住み分け

既存ツールと LKST は情報収集の密度・精度が異なる。その特徴を分類すると以下のようになる。

	既存ツール	LKST
採取方法	サンプリングベース	イベントベース
採取情報	採取要求時点のカーネル情報	イベント発生時のカーネル情報
利点	オーバヘッドが比較的少ない 容易に取得可能 採取してすぐに情報の参照が可能	情報採取の粒度は細かい イベントの実行遷移トレースを再現可能
欠点	サンプリングを細かくするとオーバヘッドが大きい 粒度が粗い 傾向を見る程度の情報量	トレース量によってはオーバヘッドが大きい 何を解析するかで採取情報が異なるため、スキルも必要

これらの特徴を活かす形で住み分けを行なう。すなわち、以下のように実施する。

従来ツール
- (1) 従来ツールによりカーネルの各機能（CPU,、メモリ、ネットワーク等）の負荷傾向を得る。
- (2) (1) の情報を元に、不自然な負荷変化のある場所や、不安定なカーネル処理を起こしている場所を検出する。

LKST
- (3) (2) の情報を元に、LKST でトレースを取る時間帯、負荷状態、採取イベントの絞込みを行なう。
- (4) (3) で指定した条件で LKST を実行し、より細かい情報を収集する。
- (5) (4) を回収し、多面的に解析する。

1.6 構成

lkstlogtools は以下のようなソフトウェア・パッチで構成される。

- LKST を適用したカーネルに適用する LKST 機能拡張パッチ
- LKST パッケージ自体に適用する LKST ツール拡張パッチ
- ログの解析を行う解析ツール
- 解析データを可視化するプロットツール
- 解析の補助を行う補助ツール

1.7 インストール手順

1.7.1 カーネル再構築とインストール

LKST の構築方法は以下の通り。

(1) 下記のコマンドを実行し、スーパーユーザでログインし直す。

```
# su
  Password: （スーパーユーザのパスワードを要求されるので、ここで入力する。）
```

(2) LKST 本体のソースパッケージを展開する。ソースアーカイブは圧縮されているので、コマンドラインから下記のコマンドを実行し、ソースファイルを展開する。

```
# cd /usr/local/src
# tar xzvf lkst-2.2.1.tar.gz
```

(3) LKST 本体のディレクトリの直下に移動し、今回開発したプログラムのパッケージを展開する。

```
# cd /usr/local/src
# tar xzvf lkstlogtools.tar.gz -C lkst-2.2.1/
```

(4) LKST を適用したカーネルの構築を行う。

(a) 従来の LKST のパッチを Linux upstream カーネル 2.6.9 に当てる。（パッチが複数あるので、一括して実行するスクリプトを提供している。）

```
# ./scripts/enpatch.sh /usr/src/linux/linux-2.6.9
```

実行後、以下の確認を聞いてくる(質問の後ろに記載した[ ]の中がデフォルトの指定)ので、文末の強調部分の文字を入力する。

```
Apply preassigned page table patch (if not, apply vmsync patch automatically)
(Y)es/(N)o? [Y]  Y
```

Apply early boot-time tracing patch (optional) (Y)es/(N)o? [Y] N
Apply extra event set patch (optional) (Y)es/(N)o? [Y] N
Apply ignore hooks placed in inline-functions patch (optional) (Y)es/(N)o? [Y] N

(b) 現在のディレクトリ(/usr/local/src/)の下に、ディレクトリ“lkstlogtools”が作成されているので、そこへ移動する。

```
# cd lkstlogtools
```

(c) (b)を実行後、今回開発したパッチを当てる。(パッチが複数あるので、スクリプトを作成して提供している。)

```
# ./scripts/enpatch-llt.sh  /usr/src/linux-2.6.9
```

(5) Linux カーネルのディレクトリへ移動し、Configuration を設定する。

```
# cd /usr/src/linux/linux-2.6.9
# make menuconfig
```

この中で、LKST を有効にするために、文末の強調部分の値で設定を行なう。

"Loadable module support"を選択し、

"Module unloading" Y

"Processor type and features"を選択し、

"Assign page tables for non-contiguous mapped area on boot" Y

"Use register arguments" N

"Kernel hacking"を選択し、

"Kernel debugging" Y
"Magic SysRq key" Y
"Kernel Hook Support" Y
"Kernel State Tracing (LKST)" m

Save して終了する。

(6) LKST を適用した Linux upstream カーネルを再構築する。

```
# make modules bzImage  > make.log
```

再構築でエラーが表示されていないことを確認するため、以下のコマンドを実行して何も表示されないことを確認する。

```
# grep Error make.log
```

(7) LKST を適用した Linux upstream カーネルとモジュールを、所定のディレクトリにインストールする。

```
# make modules_install
# make install
```

モジュールがインストールされていることを確認するため、次のコマンドを実行する。

```
# ls /lib/modules/2.6.9-lkst221
```

```
# ls /lib/modules/2.6.9-lkst221
Build/                          modules.isapnpmap
kernel/                         modules.parportmap
modules.dep                     modules.pcimap
modules.generic_string          modules.pnpbiosmap
modules.ieee1394map             modules.usbmap
```

図 1 モジュール確認画面

また、Linux upstream カーネルがインストールされていることを確認するため、次のコマンドを実行する。

```
# ls /boot
```

出力画面に以下があることを確認する。

(a) System.map-2.6.9-lkst221
(b) initrd-2.6.9-lkst221.img
(c) vmlinux-2.6.9-lkst221

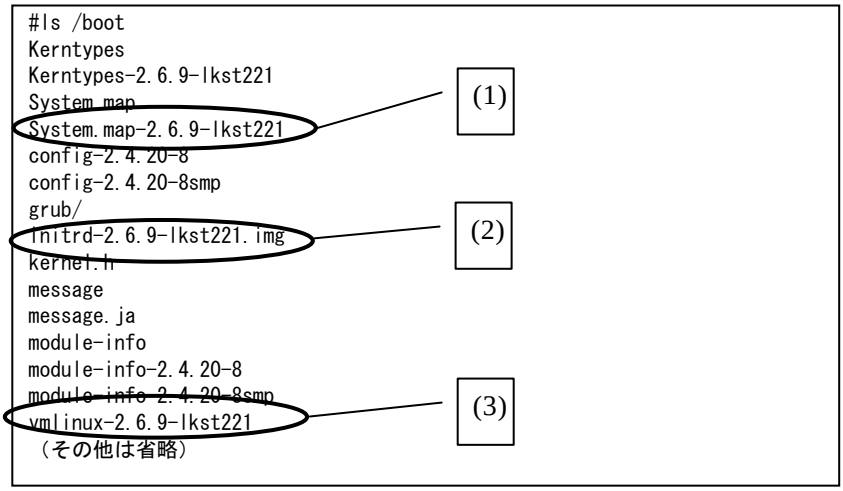


図 2 Linux upstream カーネルのインストール確認画面

(8) 起動 OS のリストに、今インストールした LKST を適用した Linux upstream カーネルが追

加されていることを確認する。

```
# cd  /boot/grub/
# cat grub.conf
```

```
# grub.conf generated by anaconda
#
(途中省略)
default=9
timeout=10
splashimage=(hd0,1)/boot/grub/splash.xpm.gz
title Original Linux (2.6.9-lkst221)
          root (hd0,1)
          kernel /boot/vmlinuz-2.6.9-lkst221 ro root=LABEL=/ hdc=ide-scsi
          initrd /boot/initrd-2.6.9-lkst221.img
title Red Hat Linux (2.6.6-lkst210)
          root (hd0,1)
          kernel /boot/vmlinuz-2.6.6-lkst210 ro root=LABEL=/ hdc=ide-scsi
          initrd /boot/initrd-2.6.6-lkst210.img
(以後省略)
```

図 3　grub.conf 内確認画面

(9) インストールした LKST を適用した Linux upstream カーネルでシステムを再起動する。再起動後、OS の確認をする。

```
#  uname -r
```

バージョン文字列が 2.6.9-lkst221 であることを確認する。

```
# uname -r
2.6.9-lkst221
```

図 4 Linux　OS　確認画面

1.7.2　LKST コマンドと解析ツールの構築とインストール

(1) LKST 本体のディレクトリの直下に移動し、従来の LKST コマンド（プログラム）に、今回開発の解析ツールに対応する情報を処理するためのパッチを当てる。

```
# cd /usr/local/src/lkst-2.2.1/
# patch -p1 < lkstlogtools/patches/lkstutils-logformat.patch
```

(2) LKST コマンドの構築のために、カーネルのソースコード（ヘッダファイル）へのパス設定を行う。

```
# cd /usr/local/src/lkst-2.2.1/
# make config
```

(3) LKST コマンドの構築とインストールを行う。

```
# cd lkstutils
# make
# make install
```

(4) 今回開発の解析ツールの構築とインストールを行う。

```
# cd /usr/local/src/lkst-2.2.1/lkstlogtools
# make
# make  install
```

ソフトウェア環境を確認する。

```
# ls /usr/sbin/lkst*
```

LKST コマンド群が表示されることを確認する。

```
#ls /use/sbin/lkst*
/usr/sbin/lkst                  /usr/sbin/lkstbuf
/usr/sbin/lksteh                /usr/sbin/lkst_fmt_sysinfo
/usr/sbin/lkstla                /usr/sbin/lkstlogd
/usr/sbin/lkstlogdiv            /usr/sbin/lkstlogger
/usr/sbin/lkstm                 /usr/sbin/lkst_make_mask
/usr/sbin/lkst_plot_dist        /usr/sbin/lkst_plot_log
/usr/sbin/lkst_plot_stat        /usr/sbin/lkst_plot_sysinfo
```

図 5　LKST コマンド及び解析ツールの確認

```
# ls modules
```

今回開発したイベントハンドラ群が表示されることを確認する。

```
# ls  modules
lksteh_procstat.ko    lksteh_sysinfo.ko
lksteh_vminfo.ko      lksteh_wqcounter.ko
```

図 6　イベントハンドラの確認

1.7.3 RPM パッケージからのインストール

MIRACLE LINUX V3.0 に今回開発したプログラムをインストールする手順は次の通り。

(1) 下記のコマンドを実行し、スーパーユーザでログインし直す。

```
# su
  Password: （スーパーユーザのパスワードを要求されるので、ここで入力する。）
```

(2) 今回開発したプログラムの kernel 2.4 用のパッチが適用された MIRACLE LINUX V3.0 用のカーネルパッケージを展開する。

```
# cd /usr/src/rpm/SRPMS
# rpm -ihv kernel-2.4.21-20.19AX.lkst1.src.rpm
```

(3) 今回開発したプログラムのカーネル用のパッチが含まれていることを確認する。

```
# cd /usr/src/rpm/SOURCES
# ls linux-2.4.21-lkstlogtools.patch
# cd /usr/src/rpm/SPECS
# grep linux-2.4.21-lkstlogtools.patch kernel-2.4.spec
Patch32100: linux-2.4.21-lkstlogtools.patch
```

(4) LKST を適用したカーネルパッケージの再構築を行う。

```
# cd /usr/src/rpm/SPECS
# rpmbuild -ba -target i686 kernel-2.4.spec
```

(5) カーネルパッケージが構築されたことを確認する。

```
# cd /usr/src/rpm/RPMS/i686
# ls
kernel-smp-2.4.21-20.19AX.lkst1.i686.rpm
kernel-source-2.4.21-20.19AX.lkst1.i686.rpm
```

(6) 構築したカーネルパッケージをインストールする。

```
# rpm -ihv kernel-smp-2.4.21-20.19AX.lkst1.i686.rpm
```

モジュールがインストールされていることを確認するため、次のコマンドを実行する。

```
# ls /lib/modules/2.4.21-20.19AX.lkst1smp
```

```
# ls /lib/modules/2.4.21-20.19AX.lkst1smp
build        modules.generic_string  modules.parportmap  modules.usbmap
kernel       modules.ieee1394map     modules.pcimap
modules.dep  modules.isapnpmap       modules.pnpbiosmap
```

図 7 モジュール確認画面

また、Linux カーネルがインストールされていることを確認するため、次のコマンドを実行する。

```
# ls /boot
```

出力画面に以下があることを確認する。

① System.map-2.4.21-20.19AX.lkst1smp

② vmlinuz-2.4.21-20.19AX.lkst1smp

```
#ls /boot
System.map
System.map-2.4.21-20.19AX.lkst1smp          ①
config-2.4.21-20.19AXsmp
grub/
initrd-2.4.21-20.19AX.lkst1smp.img
module-info
vmlinuz-2.4.21-20.19AX.lkst1smp              ②
（その他は省略）
```

図 8　Linux カーネルのインストール確認画面

(7) 起動 OS のリストに、今インストールした LKST を適用した Linux カーネルが追加されていることを確認する。

```
# cd  /boot/grub/
# cat grub.conf
```

```
# grub.conf generated by anaconda
#
(途中省略)
default=0
timeout=10
title Asianux (2.4.21-20.19AX.lkst1smp)
        root (hd0,0)
        kernel /boot/vmlinuz-2.4.21-20.19AX.lkst1smp ro root=LABEL=/
        initrd /boot/initrd-2.4.21-20.19AX.lkst1smp.img
title Asianux (2.4.21-9.30AXsmp)
        root (hd0,0)
        kernel /boot/vmlinuz-2.4.21-9.30AXsmp ro root=LABEL=/
        initrd /boot/initrd-2.4.21-9.30AXsmp.img (以後省略)
```

図 9 grub.conf 内確認画面

(8) 今インストールしたLKSTを適用したLinuxカーネルでシステムを再起動し、再起動後、OS の確認をする。

```
#  uname -r
```

バージョン文字列が 2.4.21-20.19AX.lkst1smp であることを確認する。

図 10 Linux OS 確認画面

(9) 続いて、LKST コマンドと解析ツールを次の手順で、SRPM パッケージから構築・インストールする。

i. /usr/src/rpm/SRPMS に、今回開発した LKST コマンドと解析ツールのソースを含む lkstutils-2.2.1-11AX.src.rpm があることを確認する。

```
# cd /usr/src/rpm/SRPMS
# ls lkstutils-2.2.1-11AX.src.rpm
```

ii. lkstutils のソースパッケージを展開する。

```
# rpm -ihv lkstutils-2.2.1-11AX.src.rpm
```

iii. さきほど作成した kernel-source パッケージをシステムにインストールする。

```
# rpm -ihv kernel-source-2.4.21-20.19AX.lkst1.i686.rpm
```

iv. lkstutils パッケージの構築とインストールを行う。

```
# cd /usr/src/rpm/SPECS
# rpmbuild -ba lkstutils.spec
# cd /usr/src/rpm/RPMS/i386
# rpm -Uhv lkstutils-2.2.1-11AX.i386.rpm
```

v. ソフトウェア環境を確認する。

```
# ls /usr/sbin/lkst*
```

LKST コマンド群が表示されることを確認する。

```
#ls /use/sbin/lkst*
/usr/sbin/lkst                  /usr/sbin/lkstbuf
/usr/sbin/lksteh                /usr/sbin/lkst_fmt_sysinfo
/usr/sbin/lkstla                /usr/sbin/lkstlogd
/usr/sbin/lkstlogdiv            /usr/sbin/lkstlogger
/usr/sbin/lkstm                 /usr/sbin/lkst_make_mask
/usr/sbin/lkst_plot_dist        /usr/sbin/lkst_plot_log
/usr/sbin/lkst_plot_stat        /usr/sbin/lkst_plot_sysinfo
```

図 11 LKST コマンド及び解析ツールの確認

```
# ls /lib/modules/2.4.21-20.19AX.lkst1smp/kernel/drivers/lkst/
```

今回開発したイベントハンドラ群が表示されることを確認する。

```
# ls /lib/moduels/2.4.21-20.19AX.lkst1smp/kernel/drivers/lkst
lksteh_procstat.o    lksteh_sysinfo.o
lksteh_vminfo.o      lksteh_wqcounter.o
```

図 12 イベントハンドラの確認

2. LKST 使用方法

LKST の使用方法については従来の方法と変わらない。サマリを以下で説明する。

2.1 提供コマンド一覧

各コマンドの詳細については、LKST のパッケージに付属するオンラインマニュアルを参考。

(1) LKST の状態表示

コマンド名：lkst

(2) LKST のマスクセット操作

コマンド名：lkstm

(3) LKST のバッファ操作

コマンド名：lkstbuf

(4) LKST のイベントハンドラ操作

コマンド名：lksteh

2.2 起動と終了

(1) LKST のロードとアンロード

LKST を正常にコンパイルしインストールした場合、以下の手順で LKST のドライバモジュールを組み込むことが可能である。

```
#/sbin/modprobe lkst
```

以下のコマンドを入力し、lkst が実行していることを確認する。

```
# lkst status
```

以下の画面が表示されれば LKST は稼動中である。

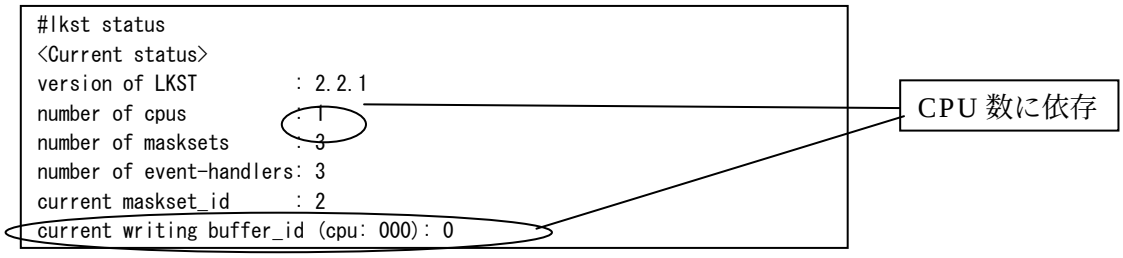


図 13 LKST 稼動状況の確認

ドライバモジュールをカーネルからはずしたい場合は、以下のようにする。

```
# /sbin/rmmod lkst
```

(2) LKST の状態確認

LKST に関するすべての状態を知るには、lkst コマンドを使い、以下の手順で状態を表示させる。

```
# lkst all
```

(3) LKST の有効化と無効化

LKST の有効化：

LKST はドライバをロードした時点で有効化されており、デフォルトでは 44 個のイベントをトレースする。無効化されている場合は以下のようにして有効化する。

```
# lkst start
```

LKST の無効化：

LKST を無効化するには以下のようにする。

```
# lkst stop
```

2.3 情報取得方法と取得情報の見方について

(1) 情報の取得方法

LKST で記録した情報をカーネルから取得するには、lkstbuf コマンドを用いる。

```
# lkstbuf read -f logfile
```

上記のコマンドを実行すると、logfile で指定されたファイルに、LKST で記録した情報が保存される。より詳しくは、lkstbuf コマンドのオンラインマニュアルを参照。

(2) 取得情報の表示方法

LKST のログファイル内容を表示するのにも、lkstbuf コマンドを用いる。

```
#lkstbuf print -f logfile
```

上記のコマンドを実行すると、logfile で指定されたログファイルの内容を、テキスト形式で表示する。より詳しくは、lkstbuf コマンドのオンラインマニュアルを参照。

2.4 取得する情報の変更

LKST で記録するイベントを変更するには、LKST のマスクセット内容を変更する。LKST は、カーネル内部で発生するイベントと、それぞれのイベントをどのように処理するか、という組み合わせをマスクセットという組み合わせ表を利用して管理している。LKST はマスクセットを最大 254 個持つことが出来る。それぞれのマスクセットはマスクセット番号を持っており、lkstm コマンドでマスクセット番号を指定することで、そのマスクセットを有効にすることが出来る。

```
# lkstm set -m 1
```

上記のコマンドでマスクセット番号１番のマスクセットを有効にすることが出来る。

マスクセットの内容を取得したい場合は、以下のコマンドを実行する。

```
# lkstm read -m 2 -f <masksetfile>
```

上記のコマンドを実行すると、masksetfile で指定されたファイルに、マスクセット番号 2 のマスクセットの内容が書き込まれる。

また、この masksetfile に書かれた組み合わせ表を書き換えることで、それぞれのイベントに対してどういった処理をするかを設定することが出来る。

```
# lkstm write -m 5 -f <masksetfile>
```

上記のコマンドを実行することで、masksetfile の内容に従い、マスクセット番号 5 のマスクセットを新規に作成、あるいは修正することが出来る。ただするクセット番号が 0,1,2 の 3 つは、LKST が内部に持っており、これらの内容を書き換えることは出来ない。

詳しくは lkstm のオンラインマニュアルを参照。

2.5 取得できる情報量の変更

LKST で取得できる情報量を変更するには、lkstbuf コマンドを用いて LKST の内部のバッファの作成・削除を行う。

```
# lkstbuf create -s 10M
```

上記のコマンドを実行することで、新たに 10MB のバッファが作成される。現在持っているバッファの一覧は、以下のコマンドを実行して調べることが出来る。

```
# lkstbuf ls
```

新規作成したバッファは、そのままでは利用できない。それぞれのバッファは ID をもっており、これを指定して以下のコマンドを実行することで新規に作成したバッファを利用することが出来る。

```
# lkstbuf jump -b <ID>
```

詳しくは lkstbuf のオンラインマニュアルを参照。

3. マニュアル

3.1 解析ツールのマニュアル

3.1.1 概要

lkstla(LKST Log Analyzer)は LKST で取得したログデータを解析し、性能評価を行うためのデータを得るコマンドである。現在 20 種類のログデータ解析器と、3 通りの表示形式をサポートしている。

3.1.2 コマンドライン文法

lkstla 〈analyzer〉 [format] [option(s)] logfile(s)

3.1.3 使用説明

analyzer には解析器（アナライザ）を指定する。20 種類の解析器を用いることで、状況に応じて柔軟な解析が可能である。アナライザはログデータを解析し、解析できたイベントごとに「キー」「デルタ値」「イベント開始時間」「PID」を返す。アナライザには 2 種類あり、2 つのログイベントの対を 1 つの解析イベントとして出力するセッション型アナライザ（例えばシステムコールアナライザなど）と、ログイベント 1 つに対して 1 つの解析イベントとして出力するイベント型アナライザがある。上記の処理結果内容や意味については、アナライザに依存する。詳しくは「3.1.5 解析器（アナライザ）一覧」を参照。

format には、解析結果の表示形式を指定する。3 通りの表示形式（フォーマット）により、最適な方法での結果の表示が可能である。表示はすべて標準出力に出力される。詳しくは「3.1.6 表示形式（フォーマット）」を参照。

option(s)には基本オプションを指定する。基本オプションではいくつかのイベントフィルタを提供している。これらのフィルタを用いて、必要な情報だけを解析することが可能である。詳しくは「3.1.4 基本オプション」を参照。

logfile(s)には解析を行うログデータの入った入力ファイルを指定する。入力ファイルの形式は、lkst-2.2.1 の lkstutils にパッチを当てた後の、lkstbuf コマンドあるいは lkstlogd コマンドが出力する、バイナリデータ形式である。

3.1.4 基本オプション

項目	説明
オプション名	イベントキーフィルタ
文法	-k<infokey>[,…]
説明	infokey で指定されたキーのイベントだけを表示する。キーの定義はアナライザに依存する。キーは“,”（カンマ）で区切って渡すことで、複数指定することが可能である。

項目	説明
オプション名	プロセス ID フィルタ
文法	-p<PID>[,…]
説明	PID で指定されたプロセス ID のイベントだけを表示する。PID の定義は、基本的にはログデータを記録した時のカレントプロセスのプロセス ID である。ブロック IO など、処理の終了が同じプロセス内で終わらないものについては、アナライザに依存する。PID は“,”（カンマ）で区切って渡すことで、複数指定することが可能である。

項目	説明
オプション名	時間域フィルタ
文法	-t[!]<starttime>[:<endtime>|+][,…]
説明	starttime で指定された時刻以降かつ、endtime で指定された時刻あるいは starttime+span で指定された時刻までに発生したイベントだけを表示する。 starttime の前に“！”をつけた場合は、その時間帯を除く時刻のイベントだけを表示する。-t1:2 と指定した場合は、発生時刻が 1 秒～2 秒までのイベントを表示するが、-t1+2 と指定した場合は、発生時刻が 1 秒～3 秒（1+2）までのイベントを表示する。時間帯は“,”（カンマ）で区切って渡すことで、複数指定することが可能である。“,”はすべて OR として解釈されるため、“！”を使うときは注意が必要である。例えば-t1:2,3+1 では時刻 1 秒～2 秒と 3 秒～4 秒までのイベントを表示するが、-t!1:2,3+1 は-t3+1 と同じ意味になり、-t!1:2,!3+1 はすべてのイベントを表示することになる。

項目	説明
オプション名	ヘルプ表示
文法	-h
説明	lkstla のヘルプを表示する。

3.1.5 解析器（アナライザ）一覧

(1) ブロック IO 関連アナライザ

項目	説明
アナライザ名	blkqueue
説明	リクエストキューごとのブロック IO キューの長さを解析する。
キー	リクエストキューアドレス＋read/write フラグ（最下位ビットが 1 なら write,0 なら read）
エイリアス名	0x<リクエストキューアドレス>[RD/WR] RD は read、WR は write であることを表す。
デルタ値	ブロック IO キューの長さ
利用イベント	BLK_GET_REQ, BLK_PUT_REQ
プロセス ID	イベント発生時のプロセス ID
オプション	なし

項目	説明
アナライザ名	biotime
説明	ブロックデバイスごとのブロック IO 要求処理時間を解析する。
キー	ブロックデバイス番号＋read/write フラグ（最上位ビットが 1 なら write, 0 なら read）
エイリアス名	<ブロックデバイス番号>-[RD/WR]
デルタ値	ブロック IO 要求処理時間（秒）
利用イベント	BIO_MAKE_REQ, BIO_END_IO
プロセス ID	BIO_MAKE_REQ 発生時のプロセス ID
オプション	なし

項目	説明
アナライザ名	bioqueue
説明	リクエストキューの長さごとのブロック IO 要求処理時間を解析する。
キー	リクエストキュー長＋read/write フラグ（最上位ビットが 1 なら write, 0 なら read）
エイリアス名	<リクエストキュー長>-[RD/WR]
デルタ値	ブロック IO 要求処理時間（秒）
利用イベント	BIO_MAKE_REQ, BIO_END_IO, BLK_GET_REQ, BLK_PUT_REQ
プロセス ID	BIO_MAKE_REQ 発生時のプロセス ID
オプション	なし

(2) スピンロック関連アナライザ

項目	説明
アナライザ名	busywait
説明	スピンロックの呼び出し位置と、ビジーウェイト時間との関係を解析する。
キー	スピンロック呼び出し位置のアドレス値
エイリアス名	スピンロック呼び出し位置
デルタ値	ビジーウェイト時間（秒）
利用イベント	LK_SPINLOCK
プロセス ID	LK_SPINLOCK 発生時のプロセス ID
オプション	-S[ksyms] ksyms で指定したシンボルファイルから関数シンボルを得る。通常は測定している OS の/proc/kallsyms を指定する。

項目	説明
アナライザ名	spinlock
説明	スピンロックの呼び出し位置と、ロック時間との関係を解析する。
キー	スピンロック呼び出し位置のアドレス値
エイリアス名	スピンロック呼び出し位置
デルタ値	スピンロック時間（秒）
利用イベント	LK_SPINLOCK, LK_SPINTRYLOCK, LK_SPINUNLOCK
プロセス ID	LK_SPINLOCK, LK_SPINTRYLOCK 発生時のプロセス ID
オプション	-S[ksyms] ksyms で指定したシンボルファイルから関数シンボルを得る。通常は測定している OS の/proc/kallsyms を指定する。

(3) ネットワーク関連アナライザ

項目	説明
アナライザ名	netroute
説明	IPv4 ルーティングテーブルの検索時間を解析する。
キー	受信ルーティング検索なら 0、送信ルーティング検索なら 1
エイリアス名	受信ルーティング検索なら input、送信ルーティング検索なら output
デルタ値	検索時間（秒）
利用イベント	NET_V4RTOUT_ENTER, NET_V4RTIN_ENTER, NET_V4RTOUT_EXIT, NET_V4RTIN_EXIT,
プロセス ID	NET_V4RTOUT_ENTER, NET_V4RTIN_ENTER 発生時のプロセス ID
オプション	なし

項目	説明
アナライザ名	netsend
説明	パケットを送出するのにかかった時間を解析する。
キー	送信デバイスのアドレス
エイリアス名	送信デバイスのアドレス
デルタ値	パケット送出要求から実際に出て行くまでの時間（秒）
利用イベント	NET_PKTSEND, NET_START_XMIT
プロセス ID	NET_PKTSEND 発生時のプロセス ID
オプション	なし

(4) メモリ管理関連アナライザ

項目	説明
アナライザ名	palloc
説明	ページの確保時間を解析する。
キー	確保するページ枚数の２を底とする対数
エイリアス名	<確保ページ数>pages
デルタ値	確保にかかった時間（秒）
利用イベント	PAGE_ALLOC_ENTER, PAGE_ALLOC_EXIT
プロセス ID	PAGE_ALLOC_ENTER 発生時のプロセス ID
オプション	なし

項目	説明
アナライザ名	vmscan
説明	ページの回収方法ごとのページ回収時間を解析する。
キー	回収方法の種類（下表参照）
エイリアス名	回収方法名
デルタ値	ページ回収にかかった時間（秒）
利用イベント	SHRINK_SLAB_ENTER, SHRINK_CACHE_ENTER, PAGE_AGING_ENTER, PAGE_RECLAIM_ENTER, SHRINK_SLAB_EXIT, SHRINK_CACHE_EXIT, PAGE_AGING_EXIT, PAGE_RECLAIM_EXIT
プロセス ID	SHRINK_SLAB_ENTER, SHRINK_CACHE_ENTER, PAGE_AGING_ENTER, PAGE_RECLAIM_ENTER 発生時のプロセス ID
オプション	なし

種類	回収方法名	説明
0	shrink_slab	スラブからメモリを回収する
1	shrink_cache	inode/dentry キャッシュからメモリを回収する
2	page_aging	ページの加齢処理を行う
3	page_reclaim	ページの回収を行う

(5) プロセス関連アナライザ

<table>
<tr><th>項目</th><th>説明</th></tr>
<tr><td>アナライザ名</td><td>procstat</td></tr>
<tr><td>説明</td><td>プロセスごとの状態遷移を解析する。</td></tr>
<tr><td>キー</td><td>プロセス ID</td></tr>
<tr><td>エイリアス名</td><td>PROCESS_SCHED_ENTRY イベントが発生したときのプロセス名</td></tr>
<tr><td>デルタ値</td><td>プロセスの状態値（下表参照）
<table>
<tr><th>状態値</th><th>状態</th><th>説明</th></tr>
<tr><td>0</td><td>CALL_EXIT</td><td>終了処理を呼び出した。</td></tr>
<tr><td>1</td><td>TASK_DEAD</td><td>（現在未使用）</td></tr>
<tr><td>2</td><td>TASK_ZOMBIE</td><td>終了処理が完了した。</td></tr>
<tr><td>3</td><td>TASK_TRACED</td><td>デバッガなどにトレースされている。</td></tr>
<tr><td>4</td><td>TASK_STOPPED</td><td>一時停止状態である。</td></tr>
<tr><td>5</td><td>TASK_UNINTERRUPTIBLE</td><td>シグナル割り込み不可状態で処理完了待ち</td></tr>
<tr><td>6</td><td>TASK_INTERRUPTIBLE</td><td>シグナル割り込み可能状態で処理完了待ち</td></tr>
<tr><td>7</td><td>TASK_RUNNING(ready)</td><td>実行可能状態で CPU が空くのを待っている。</td></tr>
<tr><td>8</td><td>TASK_RUNNING(run)</td><td>CPU 上で実際に実行中である。</td></tr>
</table></td></tr>
<tr><td>利用イベント</td><td>PROCESS_CONTEXTSWITCH, PROCESS_CONTEXTSW2, PROCESS_WAKEUP2, PROCESS_FORK, PROCESS_EXIT, PROCESS_SCHED_ENTER</td></tr>
<tr><td>プロセス ID</td><td>状態を変更したプロセスのプロセス ID</td></tr>
<tr><td>オプション</td><td>-L
　プロセスの状態値一覧を表示して終了する。解析は行わない。
-P
　プロセスの名前の変わりに PID を表示する。</td></tr>
</table>

項目	説明
アナライザ名	proclive
説明	プロセスごとの生存時間を解析する。
キー	プロセス ID
エイリアス名	PROCESS_SCHED_ENTRY イベントが発生したときのプロセス名
デルタ値	プロセスの生存時間（秒）
利用イベント	PROCESS_FORK, PROCESS_EXIT
プロセス ID	PROCESS_FORK の時に生成されたプロセスの PID
オプション	なし

(6) スケジューラ関連アナライザ

項目	説明
アナライザ名	runqueue
説明	CPU ごとのランキューの長さを解析する。
キー	CPU 番号
エイリアス名	CPU-<CPU 番号>
デルタ値	ランキューに入っているプロセスの数
利用イベント	PROCESS_CONTEXTSWITCH
プロセス ID	PROCESS_CONTEXTSWITCH の時のプロセス ID(スイッチする前のプロセス)
オプション	なし

項目	説明
アナライザ名	schedule
説明	プロセスごとのスケジューリング処理時間を解析する。
キー	プロセス ID
エイリアス名	PROCESS_SCHED_ENTRY イベントが発生したときのプロセス名
デルタ値	スケジューリング処理にかかった時間
利用イベント	PROCESS_SCHED_ENTER, PROCESS_SCHED_EXIT
プロセス ID	PROCESS_SCHED_ENTER の時のプロセス ID
オプション	なし

項目	説明
アナライザ名	schedrun
説明	ランキューの長さと、スケジューリング処理時間の関係を解析する。
キー	ランキューの長さ
エイリアス名	rq-<ランキューの長さ>
デルタ値	スケジューリング処理にかかった時間
利用イベント	PROCESS_SCHED_ENTER, PROCESS_SCHED_EXIT
プロセス ID	PROCESS_SCHED_ENTER の時のプロセス ID
オプション	なし

(7) ソフトウェア IRQ 関連アナライザ

項目	説明
アナライザ名	softirq
説明	ソフトウェア IRQ の種類ごとに、割り込み要求があってから実行するまでの時間を解析する。
キー	ソフトウェア IRQ の種類（下表参照）
エイリアス名	ソフトウェア IRQ 名
デルタ値	要求から実行までの時間（秒）
利用イベント	SOFTIRQ_RAISE, SOFTIRQ_ACTION
プロセス ID	SOFTIRQ_RAISE 発生時のプロセス ID
オプション	なし

種類	ソフトウェア IRQ 名	説明
0	tasklet hi	優先的タスクレット
1	timer	タイマ
2	net_tx	ネットワーク（送信）
3	net_rx	ネットワーク（受信）
4	scsi	SCSI
5	tasklet	タスクレット

項目	説明
アナライザ名	timer
説明	ソフトウェアタイマが実行された時間と設定した時間との差を解析する。
キー	タイマで実行される関数のアドレス
エイリアス名	タイマで実行される関数名
デルタ値	タイマ実行の処理遅延時間（秒）
利用イベント	TIMER_RUN
プロセス ID	TIMER_RUN 発生時のプロセス ID
オプション	-S[ksyms] ksyms で指定したシンボルファイルから関数シンボルを得る。通常は測定している OS の/proc/kallsyms を指定する。

(8) システムコール関連アナライザ

項目	説明
アナライザ名	syscall
説明	システムコールごとのシステムコールを実行している時間を解析する。
キー	システムコール番号
エイリアス名	システムコール名
デルタ値	システムコールにかかった時間（秒）
利用イベント	SYSCALL_ENTRY,SYSCALL_EXIT, SYSCALL_SYSENTER, SYSCALL_SYSEXIT
プロセス ID	SYSCALL_ENTRY, SYSCALL_SYSENTER 発生時のプロセス ID
オプション	なし

(9) ウェイトキュー関連アナライザ

項目	説明
アナライザ名	waitqueue
説明	ウェイトキューにたまっているプロセスの数を解析する。
キー	ウェイトキューに入れる場所のアドレス
エイリアス名	ウェイトキューに入れる場所
デルタ値	ウェイトキューの長さ
利用イベント	PROCESS_CNT_WQ
プロセス ID	PROCESS_CNT_WQ 発生時のプロセス ID
オプション	-S[ksyms] ksyms で指定したシンボルファイルから関数シンボルを得る。通常は測定している OS の/proc/kallsyms を指定する。

項目	説明
アナライザ名	waittime
説明	ウェイトキューで待っていた待ち時間を解析する。
キー	ウェイトキューに入れる場所のアドレス
エイリアス名	ウェイトキューに入れる場所
デルタ値	ウェイトキューでの待ち時間（秒）
利用イベント	PROCESS_ADD_WQ, PROCESS_CNT_WQ, PROCESS_REM_WQ
プロセス ID	PROCESS_ADD_WQ, PROCESS_CNT_WQ 発生時のプロセス ID
オプション	-S[ksyms] ksyms で指定したシンボルファイルから関数シンボルを得る。通常は測定している OS の/proc/kallsyms を指定する。

(10)その他

項目	説明
アナライザ名	event
説明	記録されたイベントの種類を解析する。
キー	LKST のイベント ID
エイリアス名	LKST のイベント名
デルタ値	なし（常に０）
利用イベント	すべてのイベント
プロセス ID	イベント発生時のプロセス ID
オプション	-E[etype_list] etype_list に指定したファイルから LKST のイベントタイプの記述を読み込む。通常は/proc/lkst_etypes を指定する。

3.1.6　表示形式（フォーマット）

表示形式の指定は省くことも可能である。省いた場合は時系列表示形式が選択される。

項目	説明
オプション名	統計表示形式
文法	-s
説明	各アナライザの出力から、キー、エイリアス名、発生回数、デルタ値の平均値、最大値、最小値をテキスト形式で出力する。

項目	説明
オプション名	分布表示形式
文法	-d[max[:min[:div[:log]]]]
説明	デルタ値を基準に従ってイベントのクラス分けを行い、各々のクラスに属するイベントの数を表示する。基準値は引数で指定することが可能である。 標準では、時間型は 0～100ns, 100ns～1us, 1us～10us, 10us～100us, 100us～1ms, 1ms～10ms, 10ms～100ms, 100ms～1s, 1s～10s, 10s 以上の 10 クラスに分けられる。出力が整数型である場合は、0～9, 10～20, 20～30, 30～40, 40～50, 50～60, 60～70, 70～80, 80～90, 90 以上の 10 クラスに分けられる。 -d は、最小値(min)、最大値(max)、分割数(div)、対数表示(log)、の４つのパラメータを与えることができる。省略すると、各々のアナライザのデフォルト値が使用される。引数の与え方は次のようになる。「-d 最小値:最大値:分割数:対数表示」。-d と最小値の間に空白を入れてはならない。 例えば、「0.001 から 10 までの範囲を 50 個に分割して対数表示」する場合は、「-d0.001:10:50:1」とする。「0 から１００までの範囲を１００個に分割して線形表示」する場合は、「-d0:100:100:0」とする。

項目	説明
オプション名	時系列表示形式
文法	-l[p]
説明	各アナライザの出力から、キー、エイリアス名、発生時間、デルタ値を、時系列ログの状態でテキスト表示する。オプションとして p の場所に 0 以外の値を与えると、エイリアス名と発生時間の間にそのイベントの PID が表示される。

3.2 プロットツールのマニュアル

3.2.1 概要

lkst_plot_log, lkst_plot_stat, lkst_plot_dist, lkst_plot_sysinfo の 4 つのスクリプトは、lkstla 及び lkst_fmt_sysinfo によって解析した性能評価結果を、gnuplot によってプロットし、pdf 化した後、xpdf などのコマンドによって X ウィンドウ上に表示する。

3.2.2 コマンドライン文法

lkst_plot_log <analyzed_log> <key> [linestyle]
lkst_plot_stat <analyzed_stat>
lkst_plot_dist <analyzed_dist> <alias-name>
lkst_plot_sysinfo <analyzed_sysinfo>

3.2.3 使用説明

(1) 時系列グラフプロットツール:lkst_plot_log
lkstla で解析した時系列ログを analyzed_log に指定する。また、そのうちから表示したいものを、**キー**（最も左側の列の値）で指定する。

(2) 統計結果グラフプロットツール:lkst_plot_stat
lkstla で解析した統計結果を analyzed_stat に指定する。

(3) 対数分布ヒストグラムプロットツール:lkst_plot_dist
lkstla で解析した対数分布結果を analyzed_dist に指定する。また、そのうちから表示したいものを、**エイリアス名**（左側から 2 番目の列の値）で指定する。

(4) システム情報グラフプロットツール：lkst_plot_sysinfo
lkst_fmt_sysinfo で解析したシステム情報の時系列ログを analyzed_sysinfo に指定する。

3.3 補助ツールのマニュアル

3.3.1 概要

lkst_make_mask スクリプトは、lkstla の各アナライザに必要なマスクセットを設定する。また、lkst_fmt_sysinfo スクリプトは、LKST のログデータファイルから、システム情報を解析し、表示する。lkstlogdiv(LKST Log Divider) は、lkstlogd で取得したログデータや、SMP 構成のマシンにおいて lkstbuf で取得したログファイルを、時系列順が逆になる部分から別ファイルに分割する。

3.3.2 コマンドライン文法

lkst_make_mask 〈maskset-name〉〈analyzer〉[…]
lkst_fmt_sysinfo 〈lkstlogfile〉
lkstlogdiv 〈lkstlogfile〉

3.3.3 使用説明

(1) マスクセット作成ツール：lkst_make_mask

maskset-name で指定された名前のマスクセットを新規に作り、analyzer で指定されたアナライザに対応するイベントのイベントハンドラを適切に設定する。事前に LKST の拡張機能 lksteh_sysinfo.ko と lksteh_procstat.ko, lksteh_wqcounter.ko をロードしておく必要がある。

(2) システム情報解析ツール：lkst_fmt_sysinfo

lkstlogfile で指定された LKST のログデータファイルから、LKST の拡張機能 lksteh_sysinfo.ko で記録した情報を解析し、時系列で出力する。

(3) ログ分割ツール：lkstlogdiv

lkstlogfile には lkstlogd の出力したログファイルや、SMP マシンにおいて lkstbuf で取得したログファイルを指定する。分割後のファイルは lkstlogfile-0, lkstlogfile-1, …という名前がつけられる。

lkstlogd はログファイルがいっぱいになると、ログデータを先頭から上書きしていくため、終了時点によっては、最後に書き込んだログデータがログファイルの中間に来てしまう。また、SMP 構成のマシンで“ lkstbuf read -f xxx ”を実行し、xxx という一つのファイルに複数の CPU のバッファ内容を取り込むと、複数の CPU から読み出したデータをソートせずに連結する。この場合 CPU 数のソートされたログデータが、単純に連結された状態になる。時系列順に並んでいないログデータでは、マージソートができない。この問題を解決するため、時系列順が逆になる部分を境目にファイルを n 分割する。
またバッファの上書きが発生していた場合、LKST_OVWRTN_REC イベントが書き込まれる。LKST_OVWRTN_REC イベントの時刻は読み出した時刻であるが、このイベントの後のイベントは、イベントが発生した時刻であるため、時系列順が逆になる。LKST_

OVWRTN_REC が記録されているデータは、基本的には不正確なデータであるため、廃棄することを推奨する。例外的に、各バッファの先頭にある LKST_OVWRTN_REC イベントは無害である。これは前回読み出しを行ってから、しばらくたったために発生したものであり、今回の読み出し内容には関連しない。このツールでは、分割したバッファの先頭にある LKST_ OVWRTN_REC イベントは除去する。

4. 付録

付録 A､B において lkstlogtools の開発で使用した従来の LKST で提供されている機能拡張の方法について説明する。

4.1 付録Ａ：LKST のフックポイントの追加方法

LKST を適用したカーネルのソースコードの任意の場所に、フックポイントを挿入する方法を説明する。

4.1.1 フックポイントを挿入するファイル

フックしたい場所に、フックポイントマクロと、イベント ID、引数を書き込む。

例えば以下のコードにフックポイントを挿入したいとする。

```
int normal_function(int flag, struct sample *smpl)
{
        char *ptr=smpl->buf;
                                /*←ここに挿入したい*/
        while(*ptr != '¥0' ) {
                if (flag) {
                        …
```

イベント ID を LKST_ETYPE_SAMPLE_NORMAL というマクロであらわし、引数に flag, smpl をとるとすると、以下のようなコードになる。

```
#include <linux/lkst_hook.h>
int normal_function(int flag, struct sample *smpl)
{
        char *ptr=smpl->buf;
        LKST_HOOK(LKST_ETYPE_SAMPLE_NORMAL, LKST_ARG(flag),LKST_ARGP(smpl),
                LKST_ARG(0), LKST_ARG(0));
        while(*ptr != '¥0' ) {
                if (flag) {
                        …
```

コーディング上注意しなければならない点は、以下の４つである。

- linux/lkst_hook.h をインクルードする必要がある。
- フック場所がマクロやインライン関数、あるいはフック場所が複数ある場合は LKST_HOOK_INLINE マクロを用いなければならない。それ以外の場所では LKST_HOOK マクロを用いる。
- 引数は常に 4 つ必要である。引数があまる場合は 0 などを渡す。
- 32 ビット以下の整数の引数は LKST_ARG()マクロを使って渡す。ポインタ引数は LKST_ARGP()マクロを使う。64 ビットの引数は、そのまま渡すことが出来る。

さらにこのとき利用したイベント ID のマクロ（LKST_ETYPE_SAMPLE_NORMAL）が定義されていない場合、このイベント ID を定義する必要がある。この方法を次節で述べる。

4.1.2　イベントを定義するファイル

LKST のイベント定義用ヘッダファイルに、新たに追加したフックポイントで使用するイベントのイベント ID の値、名称、タイプ、引数の説明を記述する。arch/以下のファイル、あるいは include/asm-i386/以下のヘッダファイルなどの、アーキテクチャ依存のソースコードにフックポイントを追加した場合は、include/asm-i386/lkst_etypes.h を編集する。それ以外の場所でフックポイントを使っている場合は、include/linux/lkst_etypes.h を編集する。どちらのファイルも書き方はまったく同一である。

例えば前節で追加したイベント SAMPLE_NORMAL を定義するには以下のようにする。

```
LKST_ETYPE_DEF(0x100, NORMAL, SAMPLE_NORMAL, “a sample of normal_function”, ¥
  “flag value”, ¥
  “sample structure”, ¥
  NULL, NULL)
```

このコードでは、次のことを定義している。

- イベントの名前が SAMPLE_NORMAL であること。
- イベント ID のマクロ名が LKST_ETYPE_SAMPLE_NORMAL であり、その値が 0x100 であること。
- フックポイントはコード上に一度しか出現しない。
- イベント自体の説明は“a sample of normal_function”であること。
- 第一引数と第二引数の説明は、それぞれ、”flag value”と”sample structure”であること。
- 第三引数と第四引数は意味を持たないこと。

コーディング上注意しなければならない点は、以下の 2 点である。

- マクロコードなので終端に“；”をつけてはいけない。
- 意味のない引数には、NULL を指定する。

より詳しい情報は、lkst-2.2.1.tar.gz に含まれる Howto.txt を参照。

4.2　付録Ｂ：LKST のイベントハンドラの作成方法

LKST の各イベントにおいて呼び出されるイベントハンドラを新たに作成する方法を説明する。LKST のイベントハンドラは、カーネルドライバとして作成することが可能である。
以下にイベントハンドラプログラムのテンプレートを示す。

```
#include<linux/version.h>
#include<linux/module.h>
#include<linux/init.h>
#include<linux/lkst_private.h>

MODULE_AUTHOR(/*あなたの名前(文字列)*/);
MODULE_DESCRIPTION(/*イベントハンドラの説明(文字列)*/);
MODULE_LICENSE("GPL");/*GPL 以外は不可*/

/*イベントハンドラ本体*/
static void lkst_evhandler_example(void *phookrec, int event_type,
                                   lkst_arg_t arg1, lkst_arg_t arg2,
                                   lkst_arg_t arg3, lkst_arg_t arg4)
{
        preempt_disable(); /*重要*/
                /* ←ここにイベントハンドラの処理を書く*/
        preempt_enable();  /*非常に重要*/
}
/*sysfs インタフェース（オプショナル）*/
static ssize_t example_ctrl_store(struct lkst_eh_device *ed,
                                  const char *buf, size_t count)
{
                /* ←ここに入力文字列(buf)に対する処理を書く*/
        return count; /* 基本的には count を返す */
}
static ssize_t example_ctrl_show(struct lkst_eh_device *ed, char *buf)
{
                /*←ここに出力バッファ(buf)に対する処理を書く(sizeof(buf)<4KB)*/
        return strlen(buf) + 1; /* 書きこんだバイト数を返す */
}
static LKST_EH_DEV_DEF(/*イベントハンドラ名（文字列ではない）*/,
                       lkst_evhandler_example,
                       NULL, /*特殊コントロールインタフェース（今回は NULL）*/
                       example_ctrl_store,
                       example_ctrl_show);
/*初期化関数の設定*/
static int mod_init(void);
static void mod_cleanup(void);
module_init(mod_init);
module_exit(mod_cleanup);

―――次ページへ続く―――
```

```
―――前ページから続く―――

static int mod_init(void)
{
        int     retval;
                /*  イベントハンドラの登録 */
        retval = lkst_eh_device_register(&LKST_EH_DEV((/*イベントハンドラ名(文字列ではない)*/));
        if (retval < 0)
                goto init_err;
        return 0;
init_err:
        mod_cleanup(); /*エラーがあった場合は、ちゃんとイベントハンドラを解放する */
        return retval;
}

static void mod_cleanup(void)
{
        if (LKST_EH_DEV(/*イベントハンドラ名(文字列ではない)*/).id != LKST_EVHANDLER_ID_VOID) {
                lkst_eh_device_unregister(&LKST_EH_DEV(/*イベントハンドラ名(文字列ではない)*/));
        }
}
```

以下の点に注意してコーディングしなければならない。

- ドライバのライセンスは GPL でなければならない。
- イベントハンドラ内では、出来るだけロックを使わない。
- イベントハンドラ内では、カーネルプリエンプションは OFF にしたほうがいい。
- sysfs インタフェースでは、show の時(ユーザから読み出したとき)にはバッファサイズが渡されていない。大体 512 バイトを目安にして、それ以上書かないようにしないといけない。
- イベントハンドラの sysfs インタフェースファイルのパスは以下のようになる。
  /sys/class/lkst/**イベントハンドラ名**/ctrl
- イベントハンドラ内部で使える lkst の関数の名前は、全て lkst_evhandlerprim_で始まる。これ以外の lkst の関数は、使用しないこと。
- イベントハンドラを例外の取得に使う場合や、割り込みハンドラで使う場合は、デッドロックや二重例外の危険性に注意すること。例えば不用意なメモリアクセスはページアクセス例外を発生させる可能性があるが、例外ハンドラ内部でページアクセス例外を発生させると二重例外でシステムがハングアップする可能性がある。

4.3 付録C：計測フックポイント・イベント一覧

それぞれのアナライザがどのイベントを使うかについては、「3.1.5」の項目を参照。

(1)lkstlogtools にて定義を修正したイベント

項目	説明
イベント名	PROCESS_CONTEXTSWTCH
ID	0x001
ファイル名	kernel/sched.c
関数	context_switch()
説明	プロセス切り替えのコンテキストスイッチで発生する
引数 1	プロセス切り替え元のプロセスの task 構造体のアドレス
引数 2	プロセス切り替え先のプロセスの task 構造体のアドレス
引数 3	プロセス切り替え元のプロセスの状態
引数 4	イベント発生時のランキューにつながっているプロセスの数

項目	説明
イベント名	PROCESS_ADD_WQ
ID	0x006
ファイル名	include/linux/wait.h
関数	__add_wait_queue(), __add_wait_queue_tail()
説明	プロセスがウェイトキューに追加される直前に発生する
引数 1	ウェイトキューの wait_queue_head のアドレス
引数 2	キューに追加するプロセスの task 構造体のアドレス
引数 3	呼び出し位置のアドレス

項目	説明
イベント名	LK_SPINLOCK
ID	0x080
ファイル名	include/asm-i386/spinlock.h
関数	_raw_spin_lock()
説明	スピンロックを取得した直後に発生する
引数 1	呼び出し位置のアドレス
引数 2	ロックの spinlock_t のアドレス
引数 3	スピンロックを取得する直前の時刻（tsc）

項目	説明
イベント名	TIMER_RUN
ID	0x0a0
ファイル名	kernel/timer.c
関数	__run_timers()
説明	タイマ関数を実行する直前に発生する
引数 1	実行するタイマ関数のアドレス
引数 2	タイマ関数の引数
引数 3	タイマ要求時刻と実際の時刻との差（マイクロ秒）

(2)lkstlogtools にて新規に追加したイベント

項目	説明
イベント名	PROCESS_SCHED_ENTER
ID	0x008
ファイル名	kernel/sched.c
関数	schedule()
説明	スケジューラの入り口で発生する
引数 1	イベント発生時のプロセスの task 構造体のアドレス
引数 2	イベント発生時のランキューにつながっているプロセスの数
引数 3	イベント発生時のプロセスのプロセス名（前 8 文字）
引数 4	イベント発生時のプロセスのプロセス名（後ろ 8 文字）

項目	説明
イベント名	PROCESS_SCHED_EXIT
ID	0x009
ファイル名	kernel/sched.c
関数	schedule()
説明	スケジューラの出口で発生する 基本的にはこの直後にコンテキストスイッチが発生する。
引数 1	イベント発生時のプロセスの task 構造体のアドレス
引数 2	イベント発生時のプロセスの次に実行されるプロセスの task 構造体のアドレス

項目	説明
イベント名	PROCESS_FORK
ID	0x00a
ファイル名	kernel/fork.c
関数	do_fork()
説明	プロセスがフォークした直後に発生する
引数 1	フォークしたプロセスのプロセス ID
引数 2	フォークしたプロセスの task 構造体のアドレス
引数 3	フォークしたときに clone に渡すフラグ
引数 4	フォークしたプロセスの状態

項目	説明
イベント名	PROCESS_EXIT
ID	0x00b
ファイル名	kernel/exit.c
関数	do_exit()
説明	プロセスが終了処理（exit）をするときに発生する
引数 1	終了するプロセスのプロセス ID
引数 2	終了コード

項目	説明
イベント名	SOFTIRQ_RAISE
ID	0x018
ファイル名	include/linux/interrupt.h
関数	__raise_softirq_irqoff()
説明	ソフトウェア IRQ の実行要求で発生する
引数 1	要求されたソフトウェア IRQ の番号

項目	説明
イベント名	SOFTIRQ_ACTION
ID	0x019
ファイル名	kernel/softirq.c
関数	__do_softirq()
説明	ソフトウェア IRQ を実行するときに発生する
引数 1	実行するソフトウェア IRQ の番号

項目	説明
イベント名	PAGE_ALLOC_ENTER
ID	0x0c0
ファイル名	mm/page_alloc.c
関数	__alloc_pages()
説明	ページ割り当て処理の直前で発生する
引数 1	ページ割り当て時の条件マスク（gfp_mask）
引数 2	割り当てるページ枚数の 2 を底とする対数
引数 3	割り当て元対象のゾーンのリストのアドレス

項目	説明
イベント名	PAGE_ALLOC_EXIT
ID	0x0c1
ファイル名	mm/page_alloc.c
関数	__alloc_pages()
説明	ページ割り当て処理の直前で発生する
引数 1	ページ割り当て時の条件マスク（gfp_mask）
引数 2	割り当てるページ枚数の 2 を底とする対数
引数 3	割り当て元になったゾーンの zone 構造体のアドレス
引数 4	割り当てられたページのアドレス

項目	説明
イベント名	SHRINK_SLAB_ENTER
ID	0x0c2
ファイル名	mm/vmscan.c
関数	shrink_slab()
説明	スラブの回収処理の直前で発生する
引数 1	回収可能かどうか調べるページ数
引数 2	ページ回収時の条件マスク（gfp_mask）
引数 3	すべてのゾーンの LRU リスト上にあるページの数

項目	説明
イベント名	SHRINK_SLAB_EXIT
ID	0x0c3
ファイル名	mm/vmscan.c
関数	shrink_slab()
説明	スラブの回収処理の直後で発生する
引数 1	回収可能かどうか調べるページ数
引数 2	ページ回収時の条件マスク（gfp_mask）
引数 3	すべてのゾーンの LRU リスト上にあるページの数
引数 4	実際に調べたページの数

項目	説明
イベント名	SHRINK_CACHE_ENTER
ID	0x0c4
ファイル名	mm/vmscan.c
関数	shrink_cache()
説明	キャッシュ(inode/dentry)の回収処理の直前で発生する
引数 1	キャッシュ回収するゾーンの zone 構造体のアドレス
引数 2	scan_control 構造体のアドレス
引数 3	回収可能かどうか調べる最大ページ数

項目	説明
イベント名	SHRINK_CACHE_EXIT
ID	0x0c5
ファイル名	mm/vmscan.c
関数	shrink_cache()
説明	キャッシュ(inode/dentry)の回収処理の直後で発生する
引数 1	キャッシュ回収したゾーンの zone 構造体のアドレス
引数 2	scan_control 構造体のアドレス
引数 3	実際に調べたページ数

項目	説明
イベント名	PAGE_AGING_ENTER
ID	0x0c6
ファイル名	mm/vmscan.c
関数	refill_inactive_zone()
説明	ページの加齢処理(アクセス時間によるランク付け)の直前で発生する
引数 1	ページの加齢処理するゾーンの zone 構造体のアドレス
引数 2	scan_control 構造体のアドレス
引数 3	加齢可能かどうか調べる最大ページ数

項目	説明
イベント名	PAGE_AGING_EXIT
ID	0x0c7
ファイル名	mm/vmscan.c
関数	refill_inactive_zone()
説明	ページの加齢処理(アクセス時間によるランク付け)の直後で発生する
引数 1	ページの加齢処理するゾーンの zone 構造体のアドレス
引数 2	scan_control 構造体のアドレス
引数 3	実際に調べたページ数
引数 4	加齢処理をしたページ数

項目	説明
イベント名	PAGE_RECLAIM_ENTER
ID	0x0c8
ファイル名	mm/vmscan.c
関数	try_to_free_pages()
説明	ページの回収処理すべてを実行する直前で発生する
引数 1	ページ回収時の条件マスク（gfp_mask）
引数 2	回収するページ枚数の２を底とする対数
引数 3	回収対象のゾーンのリストのアドレス

項目	説明
イベント名	PAGE_RECLAIM_EXIT
ID	0x0c9
ファイル名	mm/vmscan.c
関数	try_to_free_pages()
説明	ページの回収処理すべてを実行した直後で発生する
引数 1	ページ回収時の条件マスク（gfp_mask）
引数 2	回収するページ枚数の２を底とする対数
引数 3	回収対象のゾーンのリストのアドレス
引数 4	実際に回収したページの数

項目	説明
イベント名	BUFFER_SUBMIT_BH
ID	0x0d0
ファイル名	fs/buffer.c
関数	submit_bh()
説明	バッファヘッドによる IO 要求の際に発生する
引数 1	バッファヘッドの buffer_head 構造体のアドレス
引数 2	バッファヘッドが示すページのアドレス
引数 3	IO 要求フラグ(下表参照)
引数 4	バッファヘッドのサイズ

bit	名称	説明
1	WRITE	書き込みフラグ。1 なら書き込み、0 なら読み込み
2	AHEAD	先読み・先書きフラグ
3	BARRIER	バリア IO、この IO 要求を追い越せないようにする
4	FAILFAST	デバイスがリトライしないようにする
5	SYNC	同期 IO のヒント

項目	説明
イベント名	BUFFER_ENDIO_BH
ID	0x0d1
ファイル名	fs/buffer.c
関数	end_bio_bh_io_sync()
説明	バッファヘッドによる IO の処理が終わった際に発生する。
引数 1	バッファヘッドの buffer_head 構造体のアドレス
引数 2	バッファヘッドが示すページのアドレス
引数 3	エラーコード
引数 4	実際に IO 処理したサイズ

項目	説明
イベント名	BIO_MAKE_REQ
ID	0x0d2
ファイル名	drivers/block/ll_rw_blk.c
関数	generic_make_request()
説明	ブロック IO 要求がキューに追加される際に発生する
引数 1	ブロック IO 要求の bio 構造体のアドレス
引数 2	ブロック IO が追加される IO リクエストキューのアドレス
引数 3	IO 要求フラグ(BUFFER_SUBMIT_BH 参照)
引数 4	ブロック IO が行われるデバイス

項目	説明
イベント名	BIO_END_IO
ID	0x0d3
ファイル名	fs/bio.c
関数	bio_endio()
説明	ブロック IO の処理が終わった際に発生する
引数 1	ブロック IO 要求の bio 構造体のアドレス
引数 2	IO を行ったサイズ
引数 3	エラーコード
引数 4	ブロック IO が行われるデバイス

項目	説明
イベント名	BLK_GET_REQ
ID	0x0d4
ファイル名	drivers/block/ll_rw_blk.c
関数	get_request()
説明	IO リクエストキューから IO リクエストを取得する際に発生する
引数 1	IO リクエストキューのアドレス
引数 2	0 なら READ, 1 なら WRITE
引数 3	IO リクエストキューのサイズ
引数 4	取得した IO リクエストの request 構造体のアドレス

項目	説明
イベント名	BLK_PUT_REQ
ID	0x0d5
ファイル名	drivers/block/ll_rw_blk.c
関数	__blk_put_request()
説明	IO リクエストキューに IO リクエストを返却する際に発生する
引数 1	IO リクエストキューのアドレス
引数 2	0 なら READ, 1 なら WRITE
引数 3	IO リクエストキューのサイズ
引数 4	返却した IO リクエストの request 構造体のアドレス

項目	説明
イベント名	NET_V4RTIN_ENTER
ID	0x0e0
ファイル名	net/ipv4/route.c
関数	ip_route_input_slow()
説明	IPv4 の受信ルーティングテーブルを検索する直前に発生する
引数 1	受信パケットの sk_buff 構造体のアドレス
引数 2	インターネットフローを表す flowi 構造体のアドレス
引数 3	受信先アドレス
引数 4	送信元アドレス

項目	説明
イベント名	NET_V4RTIN_EXIT
ID	0x0e1
ファイル名	net/ipv4/route.c
関数	ip_route_input_slow()
説明	IPv4 の受信ルーティングテーブルを検索した直後に発生する
引数 1	受信パケットの sk_buff 構造体のアドレス
引数 2	インターネットフローを表す flowi 構造体のアドレス
引数 3	エラーコード

項目	説明
イベント名	NET_V4RTOUT_ENTER
ID	0x0e2
ファイル名	net/ipv4/route.c
関数	ip_route_output_slow()
説明	IPv4 の送信ルーティングテーブルを検索する直前に発生する
引数 1	検索するルーティングテーブルの rtable 構造体のアドレス
引数 2	インターネットフローを表す flowi 構造体のアドレス
引数 3	送信先アドレス
引数 4	送信元アドレス

項目	説明
イベント名	NET_V4RTOUT_EXIT
ID	0x0e3
ファイル名	net/ipv4/route.c
関数	ip_route_output_slow()
説明	IPv4 の送信ルーティングテーブルを検索した直後に発生する
引数 1	検索するルーティングテーブルの rtable 構造体のアドレス
引数 2	インターネットフローを表す flowi 構造体のアドレス
引数 3	エラーコード

項目	説明
イベント名	NET_START_XMIT
ID	0x0e4
ファイル名	net/sched/sch_generic.c
関数	qdisc_restart()
説明	パケットを送信した直後に発生する
引数 1	送信したパケットの sk_buff 構造体のアドレス
引数 2	パケットを送信したデバイスの net_device 構造体のアドレス
引数 3	エラーコード

項目	説明
イベント名	PROCESS_CNT_WQ
ID	0xef0
ファイル名	lksteh_wqcounter.c (lkstlogtools の拡張ドライバ)
関数	lkst_evhandler_wqcounter()
説明	ウェイトキューの長さを記録する際に発生する PROCESS_ADD_WQ イベントを wqcounter ハンドラで記録するように設定すると、wqcounter ハンドラ内でウェイトキューの長さを数え、PROCESS_ADD_WQ イベントを PROCESS_CNT_WQ イベントに変換して記録する。PROCESS_ADD_WQ イベントは記録されない。
引数 1	ウェイトキューの wait_queue_head のアドレス
引数 2	キューに追加するプロセスの task 構造体のアドレス
引数 3	呼び出し位置のアドレス
引数 4	ウェイトキューに入っているプロセスの数

項目	説明
イベント名	MEM_VM_INFO
ID	0xef1
ファイル名	lksteh_vminfo.c (lkstlogtools の拡張ドライバ)
関数	lkst_evhandler_vminfo()
説明	zone の情報を記録する際に発生する SHRINK_CACHE_ENTER あるいは PAGE_AGING_ENTER イベントを vminfo ハンドラで記録するように設定すると、vminfo ハンドラ内でそのときの zone の情報を取得し、MEM_VM_INFO イベントとして記録する。同時に元のイベントも記録する。
引数 1	zone 構造体のアドレス
引数 2	zone の中のフリーページの数
引数 3	(上位 32bits)nr_scan_active（下位 32bits）nr_scan_inactive
引数 4	(上位 32bits)nr_active（下位 32bits）nr_inactive

項目	説明
イベント名	SYSTEM_INFO
ID	0xef3
ファイル名	lksteh_sysinfo.c (lkstlogtools の拡張ドライバ)
関数	lkst_evhandler_sysinfo()
説明	システムの情報を記録する際に発生する 任意のイベントを sysinfo ハンドラで記録するように設定すると、(割り込み内でない限り)sysinfo ハンドラでそのときのシステムの情報を記録する。同時に元のイベントも記録する。
引数 1	(上位 32bits)全ページ数（下位 32bits）フリーページ数
引数 2	(上位 32bits)共有ページ数(※常に 0)（下位 32bits）バッファ用ページ数
引数 3	(上位 32bits)オープンしているファイル数 (下位 32bits)未使用ファイル数(※常に 0)
引数 4	(上位 32bits)inode 数（下位 32bits)未使用 inode 数

項目	説明
イベント名	PROCESS_CONTEXTSW2
ID	0xefa
ファイル名	lksteh_procstat.c (lkstlogtools の拡張ドライバ)
関数	lkst_evhandler_procstat()
説明	procstat ハンドラで PROCESS_CONTEXTSWITCH イベントを記録するように設定すると、スイッチする先のプロセスの PID を記録するためにこのイベントが記録される 同時に PROCESS_CONTEXTSWITCH イベントも記録する。
引数 1	コンテキストスイッチ先のプロセスのプロセス ID
引数 2	コンテキストスイッチ先のプロセスの task 構造体のアドレス

項目	説明
イベント名	PROCESS_WAKEUP2
ID	0xefb
ファイル名	lksteh_procstat.c (lkstlogtools の拡張ドライバ)
関数	lkst_evhandler_procstat()
説明	procstat ハンドラで PROCESS_WAKEUP イベントを記録するように設定すると、PROCESS_WAKEUP の第一引数を、起床されるプロセスの PID を記録するように、引数を変換し、PROCESS_WAKEUP2 として記録する。 PROCESS_WAKEUP イベントは記録しない！
引数 1	起床させるプロセスのプロセス ID
引数 2	起床させるプロセスの状態のマスク
引数 3	同期的に起床するかどうか

(3)従来の LKST で定義されているイベント（そのまま利用）

項目	説明
イベント名	PROCESS_WAKEUP
ID	0x002
ファイル名	kernel/sched.c
関数	try_to_wake_up()
説明	プロセスの起床で発生する
引数 1	起床させるプロセスの task 構造体のアドレス
引数 2	起床させるプロセスの状態のマスク
引数 3	同期的に起床するかどうか

項目	説明
イベント名	PROCESS_REM_WQ
ID	0x007
ファイル名	include/linux/wait.h
関数	__remove_wait_queue()
説明	プロセスがウェイトキューから削除される直前に発生する
引数 1	ウェイトキューの wait_queue_head のアドレス
引数 2	キューから削除するプロセスの task 構造体のアドレス

項目	説明
イベント名	SYSCALL_ENTRY
ID	0x030
ファイル名	arch/i386/kernel/entry.S(i386 の場合)
関数	-
説明	システムコールの入り口で発生する
引数 1	システムコール番号

項目	説明
イベント名	SYSCALL_EXIT
ID	0x031
ファイル名	arch/i386/kernel/entry.S(i386 の場合)
関数	-
説明	システムコールの出口で発生する
引数 1	システムコール番号
引数 2	システムコールの戻り値

項目	説明
イベント名	SYSCALL_SYSENTER
ID	0x032
ファイル名	arch/i386/kernel/entry.S(i386 の場合)
関数	-
説明	システムコールの入り口(sysenter を使った場合)で発生する
引数 1	システムコール番号

項目	説明
イベント名	SYSCALL_SYSEXIT
ID	0x033
ファイル名	arch/i386/kernel/entry.S(i386 の場合)
関数	-
説明	システムコールの出口(sysenter を使った場合)で発生する
引数 1	システムコール番号
引数 2	システムコールの戻り値

項目	説明
イベント名	NET_PKTSEND
ID	0x060
ファイル名	net/core/dev.c
関数	dev_queue_xmit()
説明	パケットの送信要求で発生する
引数 1	送信パケットの sk_buff 構造体のアドレス

項目	説明
イベント名	LK_SPINTRYLOCK
ID	0x081
ファイル名	include/asm-i386/spinlock.h
関数	_raw_spin_trylock()
説明	スピンロックの取得を試行した直後に発生する
引数 1	呼び出し位置のアドレス
引数 2	ロックの spinlock_t のアドレス
引数 3	ロック取得できたら 1、失敗したら 0

項目	説明
イベント名	LK_SPINUNLOCK
ID	0x082
ファイル名	include/asm-i386/spinlock.h
関数	_raw_spin_unlock()
説明	スピンロックを解除する直前に発生する
引数 1	呼び出し位置のアドレス
引数 2	ロックの spinlock_t のアドレス

4.4 付録 D：解析ツールの使用例

4.4.1 解析の準備

(1) LKST 拡張イベントハンドラドライバを以下のようにしてロードする。

```
# /sbin/modprobe lksteh_procstat
# /sbin/modprobe lksteh_sysinfo
# /sbin/modprobe lksteh_wqcounter
```

(2) マスクセット設定用スクリプトを実行し、解析用のマスクセットを作成する。以下のコマンドを入力し、解析用マスクセット(log-mask)を作る。

```
# cd /usr/local/src/lkst-2.2.1/lkstlogtools/
# lkst_make_mask log-mask syscall palloc biotime proclive procstat runqueue sysinfo
```

(3) 先ほど作ったマスクセットを LKST で使用する。

```
# lkstm set -n log-mask
```

4.4.2 負荷生成と情報取得

(1) 情報を取得するためのバッファの生成
カーネル内に情報をためておくために 20MB のバッファをカーネル内に生成する。

```
# lkstbuf create -s 20M
```

(2) 先ほど生成したバッファを使用する。

```
# lkstbuf  jump -b 1
```

(3) バッファ内容をいったん空にする。

```
# lkstbuf  read -f /dev/null
```

(4) 解析対象となる負荷を生成する。

適切な負荷を生成するコマンド(ベンチマークなど)

(5) 記録した情報をファイルに保存する。

```
# lkstbuf  read –f lkstlogdata
```

(6) 記録情報ファイルの分割を行う。

```
# lkstlogdiv  lkstlogdata
```

4.4.3 取得した情報の解析

(a) システムコールの開始から終了までの時間の計測

分割した記録情報を解析し、システムコールの開始から終了までの時間を取得する。

```
# lkstla syscall -l lkstlogdata-* > syscall.log
# lkstla syscall -s lkstlogdata-* > syscall.stat
# lkstla syscall -d lkstlogdata-* > syscall.dist
```

(b) メモリ確保に要する時間、実行回数の計測

分割した記録情報を解析し、メモリ確保に要する時間、実行回数を取得する。

```
# lkstla palloc -l lkstlogdata-* > palloc.log
# lkstla palloc -s lkstlogdata-* > palloc.stat
# lkstla palloc -d lkstlogdata-* > palloc.dist
```

(c) プロセスの状態の変化と各状態でいる時間

分割した記録情報を解析し、プロセスの状態の変化と各状態でいる時間を取得する。

```
# lkstla procstat -l lkstlogdata-* > procstat.log
# lkstla procstat -s lkstlogdata-* > procstat.stat
# lkstla procstat -d lkstlogdata-* > procstat.dist
```

(d) IO要求から終了までの時間の計測

分割した記録情報を解析し、ＩＯ要求から終了までの時間を取得する。

```
# lkstla biotime -l lkstlogdata-* > biotime.log
# lkstla biotime -s lkstlogdata-* > biotime.stat
# lkstla biotime -d lkstlogdata-* > biotime.dist
```

(e) ユーザプロセスの生成から終了までの時間の計測

分割した記録情報を解析し、ユーザプロセスの生成から終了までの時間を取得する。

```
# lkstla proclive -l lkstlogdata-* > proclive.log
# lkstla proclive -s lkstlogdata-* > proclive.stat
# lkstla proclive -d lkstlogdata-* > proclive.dist
```

(f) プロセスのランキュー情報の計測

分割した記録情報を解析し、プロセスのランキューの長さを取得する。

```
# lkstla runqueue -l lkstlogdata-* > runqueue.log
# lkstla runqueue -s lkstlogdata-* > runqueue.stat
# lkstla runqueue -d lkstlogdata-* > runqueue.dist
```

(g) カーネルのメモリ使用量およびファイルキャッシュ情報の取得

分割した記録情報を解析し、カーネルメモリの使用状況・ファイルシステム情報を取得する。

```
# lkst_fmt_sysinfo lkstlogdata-* > sysinfo.log
```

(h) 任意のプロセスの状態遷移の抽出

任意のプロセスの状態遷移を抽出する。procstat.stat から、どの PID を調べたいかを決定し、-p オプションで指定する。例えば 8658 番の PID で起きたイベントだけを切り出したい場合は以下のようにする。

```
# lkstla procstat -p8658 -s lkstlogdata-* > procstat-8658.stat
```

(i) 任意の時間に発生した IO 要求の抽出

任意の時間に発生した I0 要求を抽出する。syscall.log から、どのシステムコールの区間を調べたいかを決定し、-t オプションで指定する。

例えば1108038465.676061557から0.014634865秒間実行されているシステムコール実行中に起きたＩＯ要求から終了までの時間を調べたい場合は以下のようにする。

```
# lkstla biotime -t1108038465.676061557+0.014634865 -s lkstlogdata-* >
biotime-trim.stat
```

4.4.4 解析結果の可視化

(a) 統計データの可視化

解析した統計データをグラフにする。例えばシステムコールの解析結果を可視化する場合、

以下のコマンドを実行する。

```
# lkst_plot_stat syscall.stat
```

この結果示されるグラフの例を以下に示す。

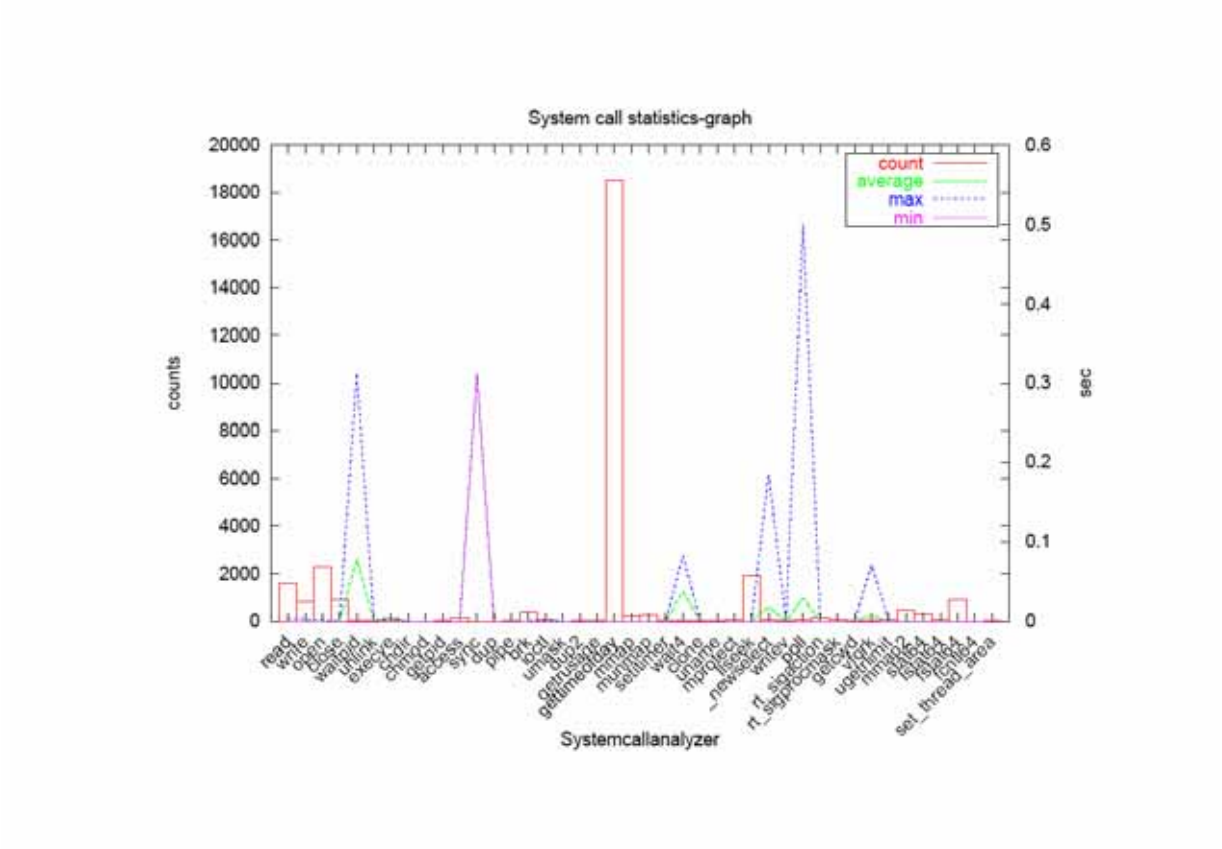


図 14 システムコール時間統計果の可視化例

(b) 分布データの可視化

解析した分布データをグラフにする。例えばシステムコールの解析結果のうち、write システムコールの処理時間分布を可視化する場合、以下のコマンドを実行する。

```
# lkst_plot_dist syscall.dist write
```

この結果示されるグラフの例を以下に示す。

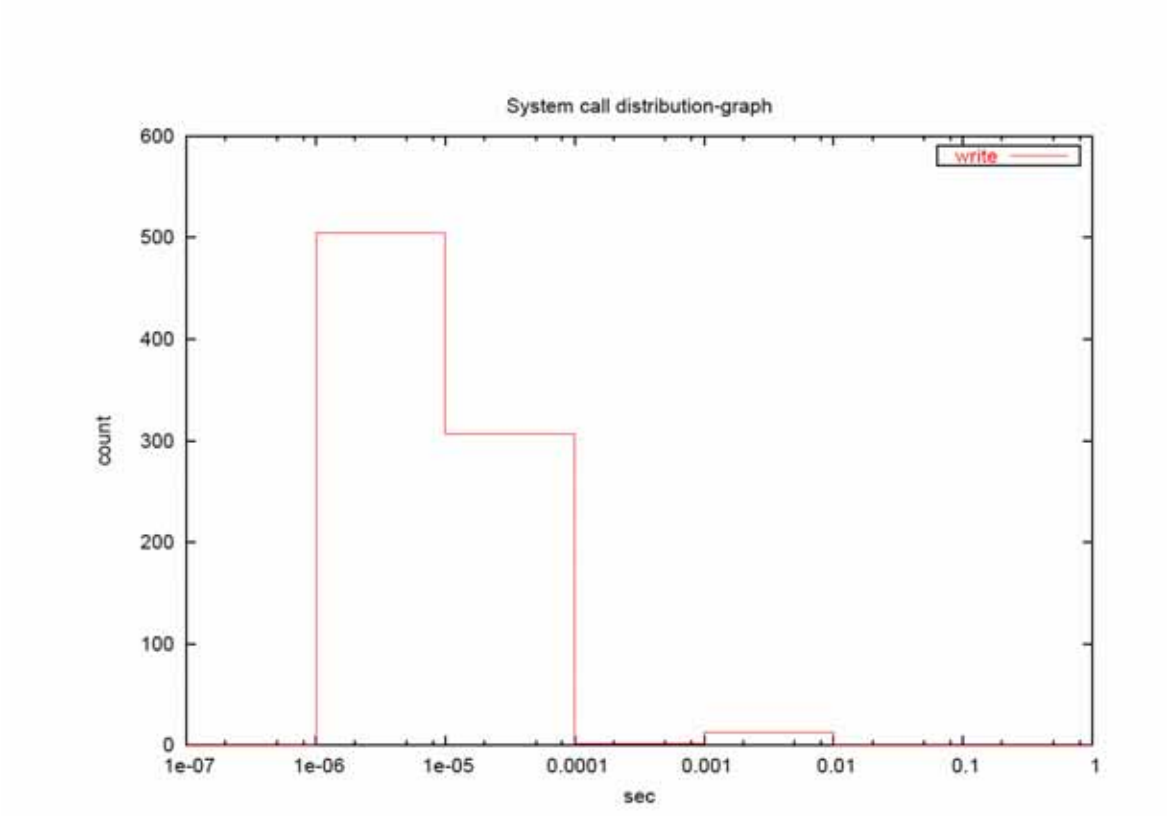


図 15 write システムコール時間分布の可視化例

(c) 時系列データの可視化

解析した時系列データをグラフにする。例えば PID=8658 のプロセスの状態遷移を可視化する場合、以下のコマンドを実行する。

```
# lkst_plot_log procstat.log 8658
```

この結果示されるグラフの例を以下に示す。

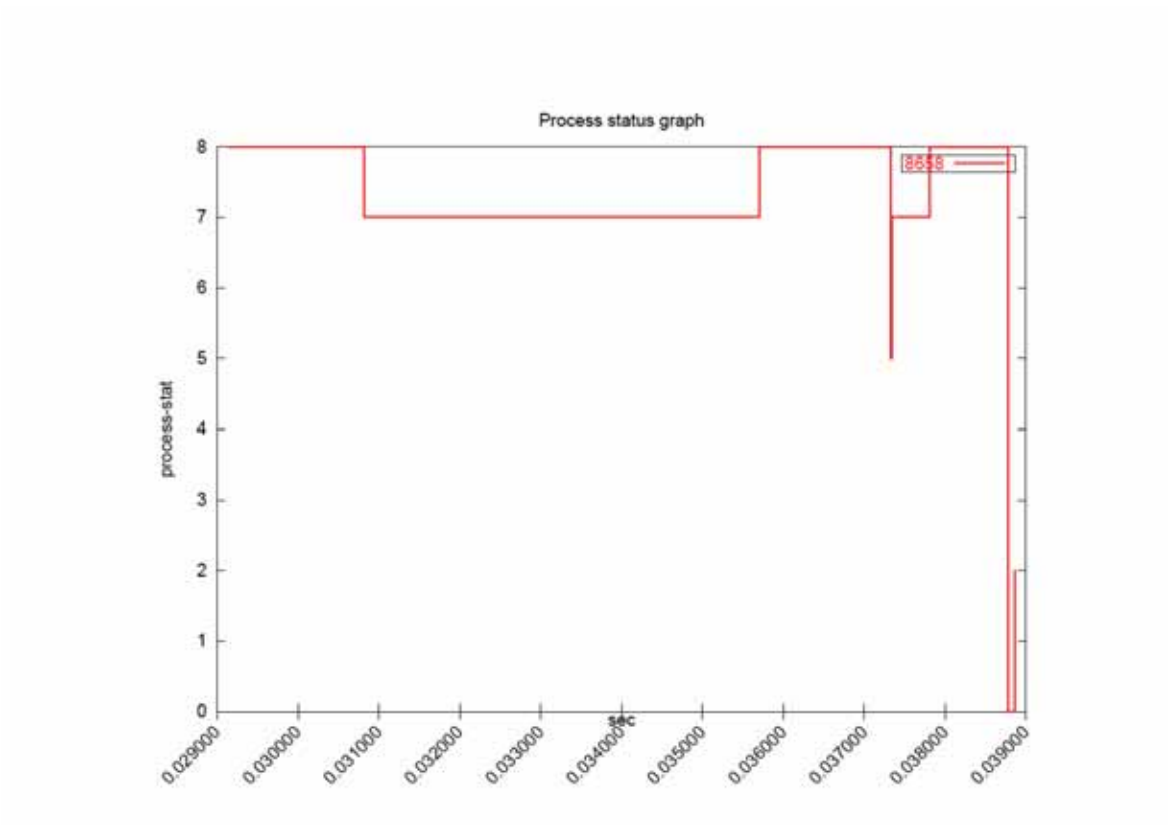


図 16 プロセス(PID:8658)状態遷移の可視化例

(d) システム情報の可視化

カーネルメモリの使用状況・ファイルシステム情報の可視化するために、以下のコマンドを実行する。

```
# ./scripts/lkst_plot_sysinfo sysinfo.log
```

この結果示されるグラフの例を以下に示す。

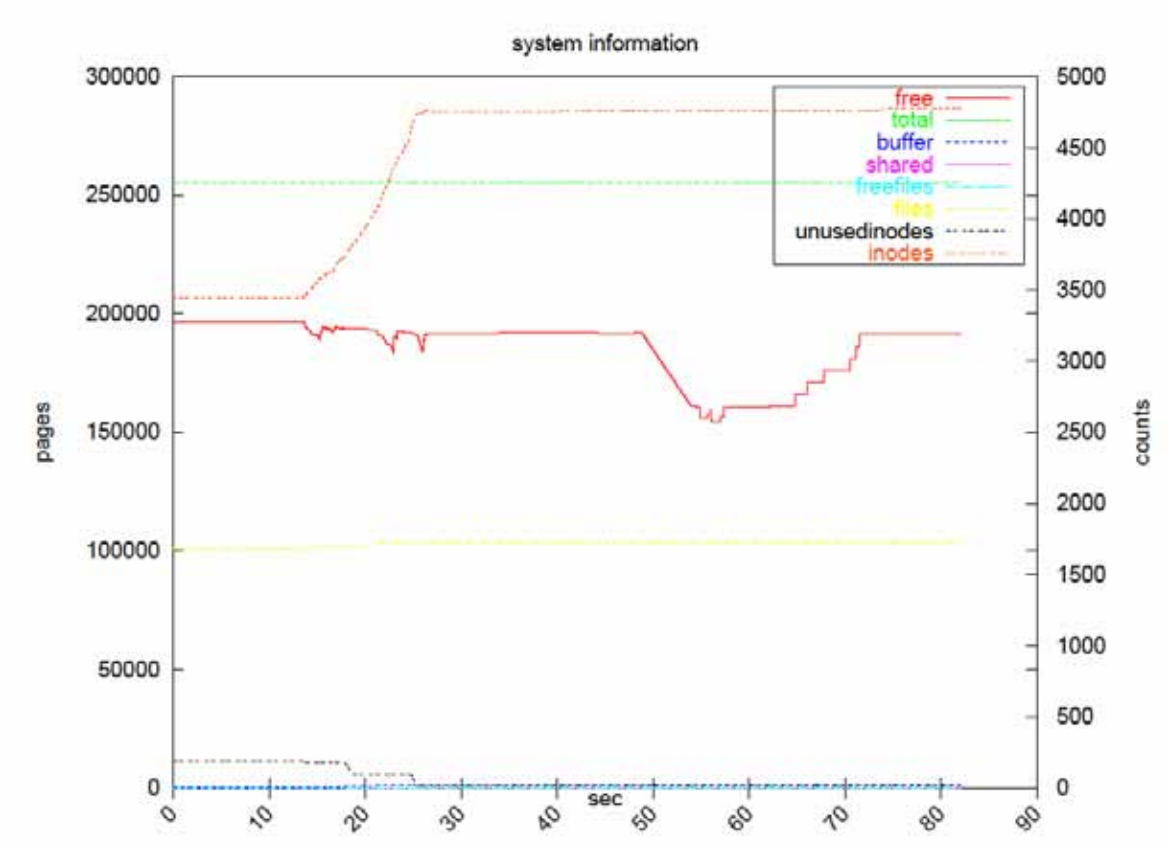


図 17 カーネル状態変化の可視化例