concurrent

RealTime Linux®: RedHawk™ アプローチ

1. 序文

コンカレントコンピュータ株式会社は、40 年以上もの間、独自 Bus から Multibus、VMEbus,PCIbus へ様々な形状のリアルタイムシステムを販売してきました。

これらの過去のリアルタイムシステムの多くは、独自のオペレーティングシステムでしたが、2000 年に完全なフリーのオープンソースである GNG/Linux をベースにしたリアルタイムプロダクトである RedHawk に軸足を移しました。

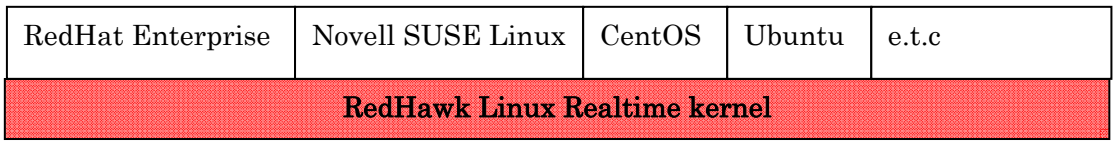
RedHawk Linux の重要な目標は、開発費を最小にするために、オープンソース・ソフトウェアのバイナリコードをリアルタイムシステムで、そのまま動作させることです。

RedHawk Linux は、Linux コミュニティで開発されたカーネルのドラスティックな変更をアプリケーションと分離するソフトウェア・レイヤー技術で、継続的に Linux カーネルの更新を可能にしています。

2. 概要

RedHawk Linux は汎用 Linux ディストリビューションではありません。

プロダクトコアには、どんなディストリビューションの基でも動作可能なリアルタイム Linux カーネルがあります。



RedHawk カーネルは、kernel.org からリリースされる安定したピュアカーネルをベースにして、ローカルに開発されたオープンソースパッチの拡張を行っています。

そして、コンカレント社は、オープンソース・コミュニティの最新リリースの修正と改良を取り入れるために、しばしば Linux カーネルベースを更新しています。

RedHawk Linux には、カーネルのリアルタイム機能へのアクセスライブラリ、ヘッダ、ドキュメンテーションとコマンド、そしてユーザレベルパッケージのセットを含まれています。

提供される API には、POSIX XSI 標準 API とコンカレントのリアルタイムソリューションを供給してきた長い歴史の間に開発された拡張 API があります。

RedHawk 1.4 based on kernel.org 2.4.21
RedHawk 2.1 based on kernel.org 2.6.3
RedHawk 2.2 based on kernel.org 2.6.7
RedHawk 2.1 based on kernel.org 2.6.3
RedHawk 2.2 based on kernel.org 2.6.7
RedHawk 2.3 based on kernel.org 2.6.9
この間欠番
RedHawk 4.1 based on kernel.org 2.6.15
RedHawk 4.2 based on kernel.org 2.6.18
RedHawk 5.1 based on kernel.org 2.6.23
RedHawk 5.2 based on kernel.org 2.6.26
RedHawk 5.3 欠番
RedHawk 5.4 based on kernel.org 2.6.31
RedHawk 6.0 based on kernel.org 2.6.36

これらの API は豊富で安定した機能とリアルタイム性を保証するインターフェースをリアルタイムアプリケーション開発者に提供します。

このインタフェースにより、コンカレントのカスタマの多くのアプリケーションが使用していたコンカレントの古い独自のソリューションから Linux カーネル 2.4 の RedHawk に移行させ、次に Linux カーネル 2.6 の RedHawk に最小限のソースコードの変更で移行することを可能にしました。

コンカレントのカスタマの多くは、合衆国軍によってまだ広く使われているＡｄａ言語で書かれたリアルタイムアプリケーションを開発し、そして実行することを望んでいます。

コンカレントの MAXAda 環境は RedHawk Linux が持つ、リアルタイム・カーネルの豊富な機能を すべて Ada 開発環境に提供します。

同様に、標準 GNU C と C++開発環境と、高性能フォートラン開発環境に対してもリアルタイム・カーネルの豊富な機能を提供します。

NightStar™ ツールと呼ばれるオプションセットの存在は、RedHawk の付加価値の多くを占めています。

NightStar ツールは、RedHawk Linux に含まれているリアルタイム・エンハンスメントを行ったカーネルの長所を最大に利用した、リアルタイム・アプリケーション開発の解析ツールとデバッキングツールを提供すると共に、リアルタイムシステムをターゲットにしたシステムチューニングツールも提供します。

NightStar ツールは、混在して記述された C, C++, Ada と FORTRAN のアプリケーションプログラムをデバッグして、分析することが出来ます。

コンカレント社では、RedHawk の価値を高めるハードウェアインテグレーション、カスタムエンジニアリングサービス、保守契約と大規模なエンドユーザドキュメンテーションサービスを行っています。

３．シールド(shield)

リアルタイムプロセスのためのシステムでリアルタイムへのコンカレントのアプローチの中核に特定の CPU の留保であるプロセッサシールドの考えがあります。

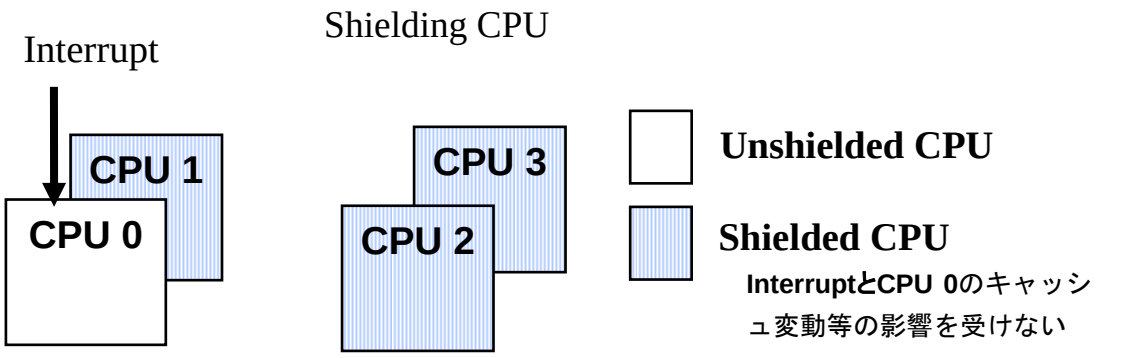
プロセッサシールドが働くためには、システムに 1 つ以上の CPU が無くてはなりません、そして 40 年の間に出荷された、ほとんどのコンカレント社のハードウェアソリューションは SMP アーキテクチャーで、しばしば洗練された NUMA システムをサポートしていました。

マルチ・コア・プロセッサが広く利用可能である今、シールドを利用することができるプラットフォームの数は爆発的に増加しました、そしてこれは高価でないシングルプロセッサコンピュータが非常に効果的なリアルタイムソリューションになることができるようになったと言う事です。

RedHawk カーネルは、プロセッサシールドの構成を設定し、そしてコントロールする

便利なユーザレベルツールを提供します。 例えば、シールドコマンド(shield)は、IRQ 割込みやローカルタイマ割込みでさえ、指定した CPU セット上のプロセスに発生するのを動的にブロックすることができます。

専用 CPU 上で走っているリアルタイムプロセスは、割込みが無いために、事実上キャッシュの干渉が無く、プロセスの決定論(determinism)が劇的に改善します、そして重要なことはプロセッサの最大性能が保証されます。他のどのようなリアルタイム機構もこのレベルの決定論(determinism)を提供することはできません。



コンカレントのプロセッサシールドの実装は、比較的小さなカーネル拡張のセットから成り立ち、(Ingo Molnar 氏によって開発された PREEMPT_RT パッチが提案している **"sleeping spinlocks**"の様な) カーネルのスピンロックと割込みモデルの全体的な再設計ではないと言う点が重要です。

この低レベル-オーバーヘッド・アプローチは、PREEMPT_RT アプローチでは到底得る事が出来ない様な、ソリューションの実現性、およびシステムの安定性と複雑性に対する影響を最小限に抑えるような劇的な強化が成されています。

"sleeping spinlocks"「眠っている スピンロック」について

"sleeping spinlocks"は、RedHat MRG が実装している PREEMPT_RT パッチの重要な部分です。

Linux カーネルの重要なリソースの排他制御するための通常の"spinlocks" は、資源を獲得する間の待ち時間にループする、「スピニング」によってシステムの性能を低下させてしまいます。

また、"spinlocks"は、リソースを得た後、そのリソースを所有している間、プリエンプションと割り込みをブロックします。

PREEMPT_RT で提供される **"sleeping spinlocks"** は、ブロックする代わりに、待っている間は、「スリープします」。 したがって、**"sleeping spinlocks"**は、リソースを獲得した後プリエンプションあるいは割り込みをブロックしません。

この設計は、高優先順位のユーザアプリケーションプロセスが早くサービスを受けることを可能にします。

しかし、PREEMPT_RT のパッチは、カーネルやデバイスドライバの spinlock_irqsave() を spinlock() に変更し、すべてのドライバの動作確認をユーザが行う必要があります。

また、割り込み処理が IRQ デーモンから起動するように変更されるため、割り込み遅延を発生させ、場合によってはシステムスループットに否定的な影響を与えます、このため、RedHawtMRG は、応答速度を保証しないソフトリアルタイム OS ですが RedHawk は、15μ秒以下の応答速度を保証するハードリアルタイム OS です。

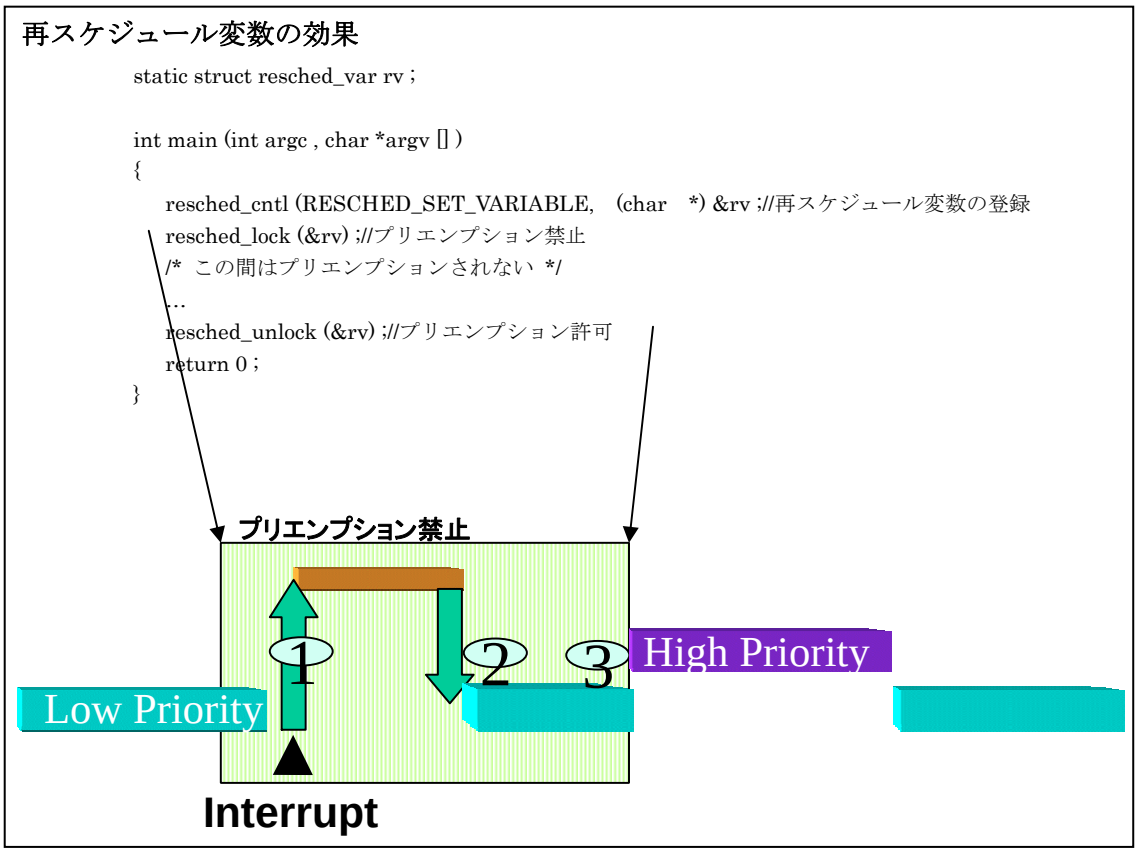
４．ユーザプリエンプション(Preemption)制御

RedHawk は実行プロセスがカーネルに、短い期間プリエンプションされない事を可能にするユーザスペース機構を提供しています。

この機能は、カーネルに組み込まれた再スケジュール変数と呼ばれる、特別なメモリ位置のプロセスレジスタを持つことによって、実現されます。

プロセスは、システムコールを行うことなく、再スケジュール変数に書くか、あるいはクリアすることによって、プリエンプションをコントロールすることができます。

この機能は RedHawk のユーザレベルライブラリ全体で使われ、ユーザレベルスピンロックの信頼できる速い実装を提供するために使われています。



５．プリアロケート・グラフィックス・ドライバ

RedHawk には、**プロセッサ間割り込み**の発生を最小化する必要から、拡張グラフィクス・ドライバが含まれています。このグラフィクス・ドライバには、オープンソースドライバとポピュラーな NVIDIA ドライバのような独自ドライバの両方が含まれています。

プロセッサ間割り込みの抑制は、主として、新しいページがグラフィクス・ドライバによってマップされるときに、TLB フラッシュの洪水が発生しないように、ブート時にグラフィクス・ページを前もって割り当てる（プリアロケーション）機構を提供することによ

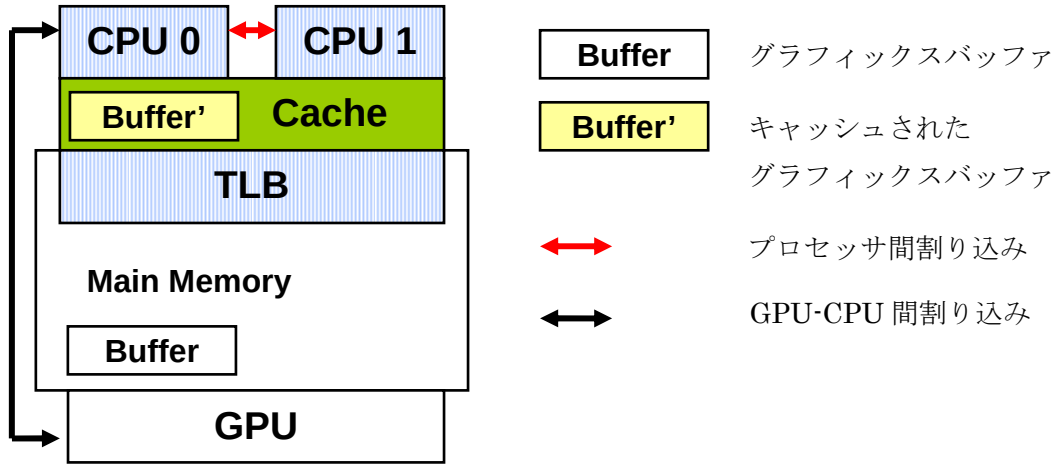
って達成されています。

例として、NVIDIA 社の CUDA 開発環境に添付されている例題プログラムが使用するメモリ量は、以下の様に 10Mバイトから 20M バイトにも達するものです。

プログラム名	ページ数(4096 バイト/ページ)
Clock	2586
SobelFilter	3458
SobolQRNG	2586
3dfd	2586
Mandelbrot	3458
oceanFFT	3714
Particles	3458
eigenvalues	2586
radixSort	2586
smokeParticles	3938
reduction	2564

これだけのメモリ量を、GPU は NVIDIA グラフィックスデバイスドライバに Linux カーネル内のバッファとして確保する様に要求します。

この要求により、NVIDIA デバイスドライバで、カーネルメモリ割り当て関数の呼び出しが行われ、結果として、マルチコア間のキャッシュの不整合と TLB バッファのフラッシュという副作用をもたらします。



これらのイベントは、**プロセッサ間割り込み**として処理され、オーバーヘッドとジッターとしてアプリケーションに影響を与えます。

実際、先の'smokeParticles' をシールドを併用して、60 秒の間 PDL(Process Dispatch Latency)試験を行うと、**プリアロケートを行わない場合**は、プロセッサ間割り込みの数は 4010 回で、PDL の最大 87μ秒でしたが、**プリアロケートを行った場合**では、プロセッサ間割り込みの数が 3 回と激減し、PDL の最大も 22μ秒になって、プリアロケートのジッター抑制効果が高いことが理解できます。

プリアロケートを行わない場合

```
Summary: PDL (60000 samples):        9.0,       12.0,       87.0 uS min/avg/max
PDL Histogram:
         0 .. 10     :     8902 ********
        11 .. 20     :    51078 **********************************************
        21 .. 30     :        4 *
        31 .. 40     :       11 *
        61 .. 70     :        3 *
        71 .. 80     :        1 *
        81 .. 90     :        1 *

プロセッサ間割り込み
retrigger_next_event_forced           1
do_flush_tlb_all                      4
__cpa_flush_range                 3,982
cache_flush    [NVIDIA]              23
```

プリアロケートを行った場合

```
Summary: PDL (60000 samples):        9.0,       12.1,       22.0 uS min/avg/max
PDL Histogram:
         0 .. 10     :     9167 ********
        11 .. 20     :    50831 **********************************************
        21 .. 30     :        2 *

プロセッサ間割り込み
retrigger_next_event_forced       3
```

６．同期(Synchronization)

RedHawk は、協調して動作するスレッドあるいはプロセス・グループ間に使われる高速な、そして効率的な sleep / wake 機構を提供しています。 このサービスは SGI 社 によって開発されたオープンソース Post/Wait のパッチに基づいています。

しかし、アトミック wakeups を保証し、決定論(determinism)を改善するために再スケジュール変数を利用したものに改良されています。

７．スケジューラ

RedHawk は、独自に開発された周期ベース・スケジューラ（FBS）と、スタンダード Linux プロセススケジューラを提供します。

FBS はユーザに周期的な実行パターンで処理行なうことができるようにする高分解能タスクスケジューラです。 FBS はオーバーラン検出を持ったメジャーとマイナサイクルを利用して多数の、調和したプロセスの周期的な実行をコントロールします。

パフォーマンスモニタ（PM）が、それぞれのスケジュールされた実行フレーム間に CPU

使用率を見るために供給されています。

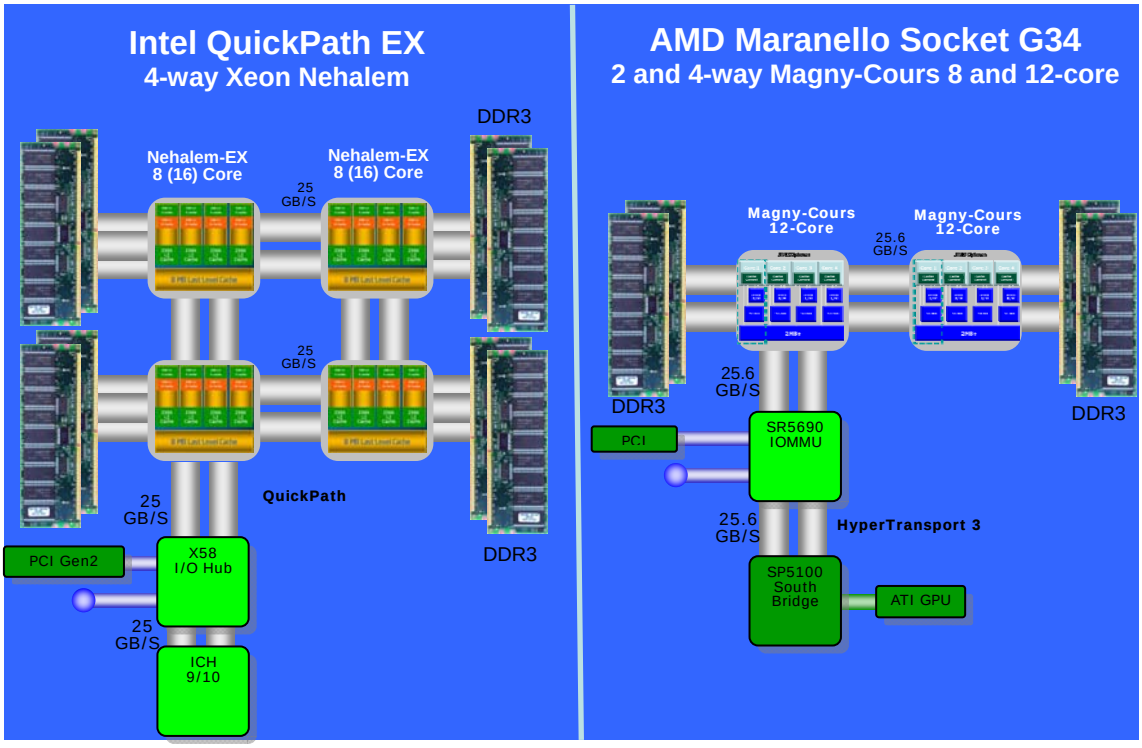
FBS インターフェースは、開発されてから数十年にわたって拡張され続け、今データ収集、シミュレーションとプロセス制御のような古典的なハード・リアルタイムアプリケーションのために最適化された理想的なスケジューラと言えるでしょう。

FBS のために書かれたアプリケーションは、ここ数年にわずかしか変更を必要としなかったのに対して、その同じ期間に Linux スケジューラはヒューリスティックな負荷バランシングと公平性スケジューラの両方の微妙な変化のために多くの書き直しを余儀なくされました。

８． NUMA (Non Uniform Memory Access)

Intel Nehalem QuickPath システムと AMD Opteron HyperTransport アーキテクチャを使う RedHawk システムでは NUMA ファシリティの全ての長所を利用することが出来ます。 コンカレントは、特定のプロセッサ（あるいはコア）と特定の NUMA ノードの両方にプロセスをバインドするための単一ツールを作成し、他のプロセス環境設定とともにメモリポリシーの指定を行えるように実行コマンドを拡張しました。

RedHawk では、所定のプロセスページのフルセットを、一つの NUMA ノード上で、ローカルであることを保証する重複ページの機能を提供することで、外部ノードからのすべてのアクセスを排除し、プロセス決定論を改善しています。



NVIDIA グラフィックカードを実装している NUMA システムでは、プリアロケートグラフィックページを指定の NUMA ノードに割り付け、性能を向上させる事が出来ます。

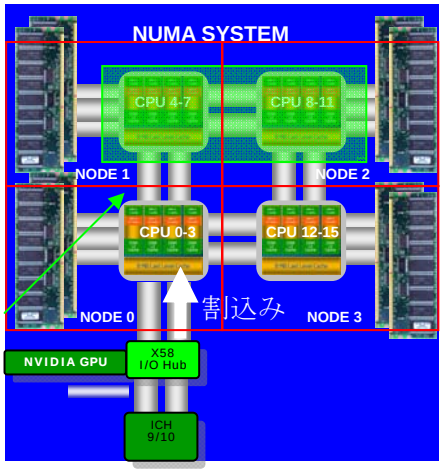
RedHawk の起動時には、ディフォルトのプリアロケートグラフィクスページは、4G バイト境界より低いアドレス位置に全ての NUMA ノードからインタリーブアクセス可能な状態で設定されていますので、これを下記の手順で再構成します。

1.最初の手順は、すべてのグラフィックプログラムを終了させて動作させるため、root で login し、init 3 を実行します。

```
# init 3
```

2.メモリシールドを ON(-m1)し、すべての割り込みから CPU4-CPU11 をシールド(-a 4-11)します。

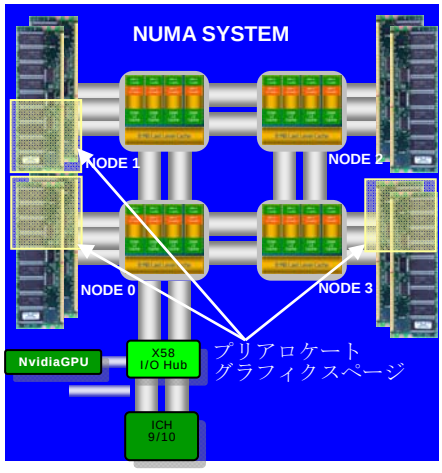
```
# /usr/bin/shield -m1 -a 4-11 -c
CPUID    irqs    ltmrs    procs    mem
-------------------------------------
0        no      no       no       no      NODE0
1        no      no       no       no
2        no      no       no       no
3        no      no       no       no
4        yes     yes      yes      yes     NODE1
5        yes     yes      yes      yes
6        yes     yes      yes      yes
7        yes     yes      yes      yes
8        yes     yes      yes      yes     NODE2
9        yes     yes      yes      yes
10       yes     yes      yes      yes
11       yes     yes      yes      yes
12       no      no       no       no      NODE3
13       no      no       no       no
14       no      no       no       no
15       no      no       no       no
```



3. /proc/driver/graphics-memory を表示し、現在の設定を確認した上で、0 を書き出して、プリアロケートグラフィックスバッファをクリアします。

(Node3 にはグラフィクスページが割り付けられていなかった事に注目してください)

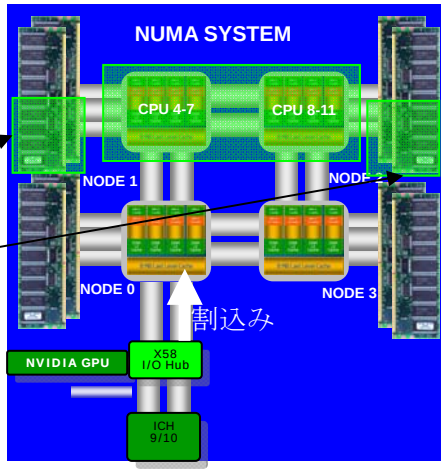
```
# /bin/cat /proc/driver/graphics-memory
Pre-allocated graphics memory:    10240 pages
Total allocated graphics memory: 10240 pages
Graphics memory in use:           0 pages
Maximum graphics memory used:     0 pages
          Node 0    Node 1    Node 2    Node 3
Preal:    5121      2712      2407      0
Total:    5121      2712      2407      0
InUse:    0         0         0         0
Max:      0         0         0         0
# /bin/echo 0   >   /proc/driver/graphics-memory
# /bin/cat   /proc/driver/graphics-memory
Pre-allocated graphics memory:    0 pages
Total allocated graphics memory:  0 pages
Graphics memory in use:           0 pages
Maximum graphics memory used:     0 pages
          Node 0    Node 1    Node 2    Node 3
Preal:    0         0         0         0
Total:    0         0         0         0
InUse:    0         0         0         0
Max:      0         0         0         0
```



4. run コマンドに bash を指定して NUMA ノード 1,2 上にグラフィクスページをインターリーブオプションで作成し確認します。
続けて、プリアロケートバッファの状態を確認し、bash を終了します。

```
# /usr/bin/run --mempolicy interleave=4,8 bash
# /usr/bin/run --mempolicy view
Mempolicy NextCpu    Cpus        Name
interleave  0x00f0    0x0ff0     run

# /bin/echo 10240 > /proc/driver/graphics-memory
# /bin/cat /proc/driver/graphics-memory
Pre-allocated graphics memory:    10240 pages
Total allocated graphics memory: 10240 pages
Graphics memory in use:           0 pages
Maximum graphics memory used:0 pages
          Node 0    Node 1    Node2    Node 3
Preal:    0         5120      5120     0
Total:    0         5120      5120     0
InUse:    0         0         0        0
Max:      0         0         0        0
# exit
```



このとき、--mempolicy interleave=4,8 は、CPU 番号指定です。
したがって、NODE1 のスタートが 4、NODE2 のスタートが 8 になり、この 2 カ所のメモリをインターリーブ（交互）に使用します。
5.最後に init 5 で X を起動して設定は終了です。

```
# init 5
```

この状態で、グラフィクスアプリケーションを、CPU 4 から CPU11 に割り付けることにより、もっとも効率良く使用できます。

また、実際の使用率は”numastat”コマンドで、確認することが出来ます。

```
$ numastat
                 node 3    node 2    node 1    node 0
numa_hit         43674     64884     79038     81643
numa_miss        0         0         0         0
numa_foreign     0         0         0         0
interleave_hit   7840      5885      4975      7015
local_node       37923     59861     75202     76404
other_node       5751      5023      3836      5239
```

numa_hit	ノードから割り当てを要求され成功したメモリの回数です。
numa_miss	ノードに割り当てることのできないため、別のノードに割り当てられたメモリ割り当て回数です。
numa_foreign	別のノードでメモリを割り当てることに失敗し、このノードから割り当てられた回数です。
interleave_hit	このノードで成功したインタリーブドメモリ割り当ての回数です。
local_node	ローカルノードに作成されたメモリ割り当て回数です。
other_node	非ローカルノードに作成されたメモリ割り当て回数です。

９．ハードウェア

コンカレントはリアルタイムソリューションが可能なコマーシャルオフザシェルフ（COTS）ハードウェアを使用することをコミットしています。

しかし不幸にも、 COTS ハードウェアの多くは、下記の理由からリアルタイムの使用に適していません。

- システム BIOS は PCI スロット IRQ 割り当ての細かい制御を許しません
- システムはディセーブル出来ないデバイスを持っていて、なおかつ PCI スロットと IRQ を共有するために物理的につながっています。
- ファームウェアが生成するディセーブル出来ない周期的な SMI 割り込み
- ハードウェアはハングしたシステムをデバッグするための、マスク不可能割込みを生成するための機構に欠けています
- チップセットによっては、リアルタイム性に対しての固有の問題を持っています。

コンカレント社によって認められたリアルタイムの条件に適合するシステムは、 iHawk™ システムと呼ばれ、リアルタイムソリューションのための公認システムリストに含められています。

評価されたシステムは、コンカレントの自動化された終夜テストシステム（ANTS）に統合されます。

出荷された iHawk システムの次の新しいバーションが出荷された場合、ボードレベル変更と新しいチップセットリビジョンがリアルタイム性能に影響を与えないことを保証するために、再度評価が行われます。

公認 iHawk COTS システムのリストには、Fujitsu, NEC, Dell, HP, Supermicro, Tyan と Newisys など多くのベンダがあります。 また、RedHawk は、種々の IBM BladeCenter ブレード環境もサポートしています。

コンカレント社のカスタマの多くは、VME ベースのシステムを運用しているため、RedHawk の VME バージョンが存在します。

コンカレント社は、PCI ホストから多くの異なったスタンダード VME I/O ボード使うことができる光ファイバーケーブルで接続する、PCI-VME バスブリッジを提供します。

さらに RedHawk はマルチコアのインテルプロセッサを走らせている数枚の異なった VME ベースのボードを選択することも出来ます。そしてこれは、VME 技術を使うことを望むカスタマが RedHawk PCI ベースのプラットフォームから２つの異なったアプローチ(ネイティブ VME あるいは VME 入出力ブリッジ)の中から選択することができます。

RedHawk は、アナログ/ディジタル変換(A/D)、デジタル/アナログ変換(D/A)、ディスクリート入出力(DIO)と、MIL-STD-1553 (航空機) や NTDS(船舶), CAN bus (自動車)等の多くのリアルタイムに最適化されたデバイス・ドライバを提供します。

コンカレント社によって出荷されたほとんどすべての iHawk システムには PCI と

PMC 両方のフォームファクタで利用可能なコンカレントで開発された Real-Time Clock & Interrupt Module (RCIM) が含まれています。

RCIM は外部装置割り込み、割り込みを発生させることができるプログラマブル割り込みジェネレータとリアルタイム時計タイマと外部コネクタを提供しています。

複数システム上で RCIM を接続し利用することで、非常に高い正度のクロックを複数マシンで同期させることができます。

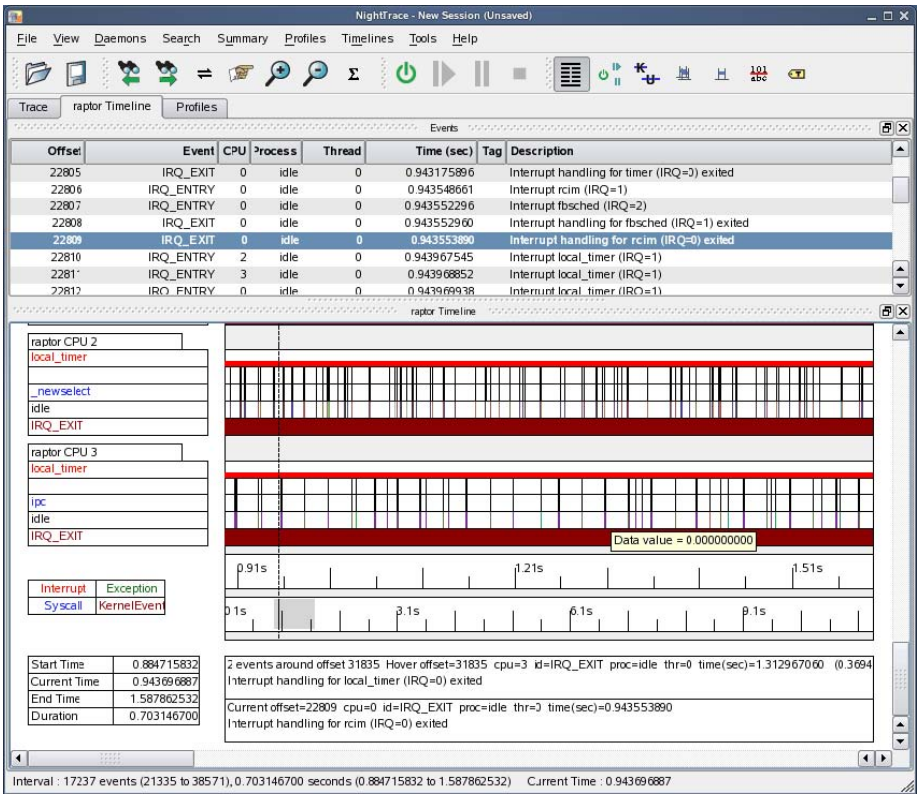
この接続によって、複数のシステムが物理的に同じシステムクロックを共有するため、RedHawk FBS を利用している同期プロセスをマスターシステムから制御することが出来るようになります。

そして RCIM はオプションとして極めて高い精度のＧＰＳ受像機と oven-controlled 水晶発振器を取り付けることができます。

１０．開発

RedHawk は、リアルタイムアプリケーションとカーネルドライバーがミスマッチしている複雑な問題を単純化するための解決方法を提供しています。

Redhawk カーネルには、リアルタイム・アプリケーションの実行を妨げないで、カーネルの動作を追跡する機構が標準で組み込まれ NightTrace™ ツールで、ユーザレベルアプリケーションとカーネルのトレースを視覚的に同時表示する事が出来ます。



ノート：Kprobes との違い

コンカレントのカーネルトレース機構は、カーネルコードトレースの鍵となる特定の位置にパーマネントトレースポイントマクロを挿入することによって実行されています。これらのマクロは条件によっては、禁止することが可能で、RedHawk はカーネルトレースの無い版が、ディフォルトで提供されています。

重要な点は、カーネルトレース機構が、リアルタイム・アプリケーションの挙動解析に使うために設計されていることです。

汎用カーネルデバッグのために特別な設計されている Kprobes のような特別なカーネルデバッグ用ツールではありません。 逆に Kprobes は多くの理由でカーネルトレースを実装するためには、貧弱な機構でしかありません。

- Kprobes は、大きなボリュームのトレースデータを効率的に集めるようには、設計されていません。またそれは INT 3 割込みを使い、スタックの大部分をデュプリケートし、そして過度の SMP 競合を起こしているユニークなハンドラーファンクションを検索するためにグローバルハッシュ表をロックします。
- Kprobes は融通がききません ; プローブを置く位置には限界があり、調査時に登録したプローブデータだけに限定されています。
- Kprobes は、特に、エンドユーザにとっては不便なツールです ; 構成を変更するたびに、カーネルモジュールのコンパイルとロードを必要とします。
- Kprobes は高レベルのメンテナンスを必要とします;進化するカーネルバーション全体に渡る正確なトレースポイントの設定を保守することは非常に難しいでしょう。

このような理由から、汎用カーネルデバッグツールとして卓越している Kprobes では、リアルタイムアプリケーションの詳細なディテールを集める事には向いていません。

NightStar ツールには、同じようにリアルタイムに最適化された NightVew デバッガと、システムチューナーNightTune が含まれています。

NightTrace と NightTune,は NightView に統合され、リアルタイムアプリケーション開発とチューニングのための特別仕立てのカスタムツールとして開発されています。

さらに、RedHawk ではカーネルドライバの開発を行うために、KDB と KGDB カーネルデバッガを組み込んでいます。

そして、カスタマサイトからクラッシュダンプを作成し、そして分析する手段として LKCD (Linux Kernel Crash Dump)メソッドの kexec / kdump をサポートしています。

１１．コミュニティ

コンカレント社は、何年もの間フリーソフトとオープンソースコミュニティで活発に活動していました。

現在コンカレント社は、run,cupid,nuu のオープンソースプロジェクトを後援し、コンカレントカーネル開発チームのメンバは、 Linux カーネルメーリングリストにバグ修正と機能のパッチをポストしています。

これまで、投稿したパッチは、高解像度タイマ、ポジックスタイマ、カーネルデバッガ、RCU 、POSIX メッセージキュー、グラフィックスドライバ、 ptrace 、 NUMA と NFS があり、いくつかのパッチは、 Linux カーネルの最新のバーションに取り入れられています。

１２．将来

現行の RedHawk6.0 は、以下のような特徴を持っています。

- エンハンスド 2.6.36 Linux カーネル
- Red Hat Enterprise Linux 6.0
- 最新の Intel と AMD マザーボードとチップセットサポート
- システム当たり 64 コア以上のサポート
- NUMA 拡張とリアルタイム性能に最適化された最新の NVIDIA グラフィックドライバー
- 最適化された CUDA 並列コンピューティング SDK
- ハードウェアとソフトウェアイベントのための Linux パフォーマンスカウンタ
- NUMA プラットフォーム用の新ページマップアドレス表示ユーティリティ
- KVM Linux カーネル仮想化サポート

しかし、RedHawk の将来のバーションには、もっとずっと多くの機能と拡張が計画されています。

これらは、コンカレントのリアルタイムユーザによって最も求められる機能から優先的に統合されていきます。

また、コンカレントは kernel.org によって受け入れられ、変更が閉じた版をコミットしています。

したがって、完全な PREEMPT_RT パッチは、十分安定し、そして強力であることを証明され kernel.org のメインストリームカーネルに受け入れられた時に適用されます。

決定性と低遅延のリアルタイム・アプリケーションを開発するための、 ベストな PREEMPT_RT をコンカレント独自に開発された機能と結合する RedHawk カーネルは、非常に強力な、そして用途が広い環境をカスタマに提供するでしょう。