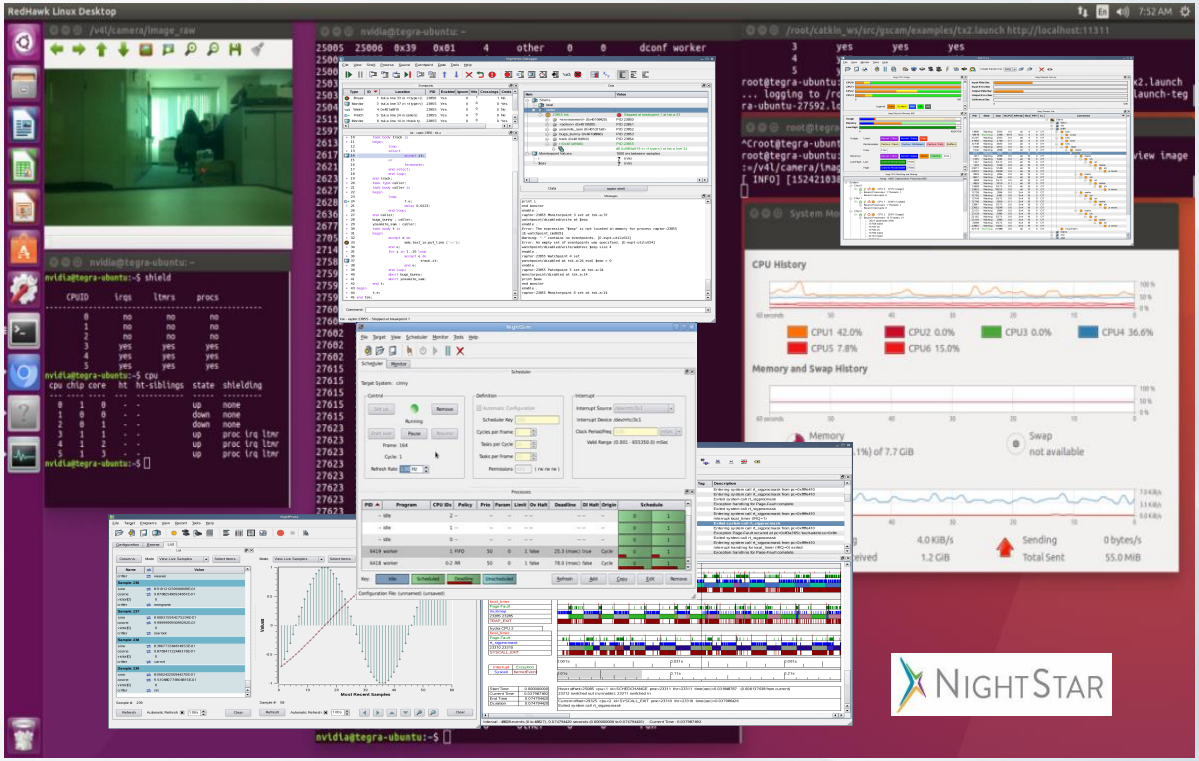
concurrent REAL-TIME

リアルタイム Linux RedHawk 7.3 for Jetson TX2/TX2iの紹介



コンカレント日本株式会社

プロフェッショナルサービス部　大島龍博

REDHAWK

第2版 2018/11/9

Concurret Real-Time社とは

- Concurret Real-Time社は1966年の創立以来50年以上にわたり、世界のリアルタイムコンピュータ技術をリードしてきた会社です。リアルタイムコンピューティングの分野において最先端の技術を 次々と開発・提供する一方で、この技術を航空/宇宙/防衛/自動車/ロボットの分野に応用し着実にその地位を不動のものにしています。
- コンカレント日本株式会社は1986年にその確かな技術と実績を 踏まえたConcurrent Real-Time社の優れた製品群を日本市場に 提供し、日本のお客様にきめ細かなサービスを行うため設立されました。以来、《お客様の求めるものを的確に理解して、それをタイムリーに提供する》ことを企業理念に掲げ、活動を続けています。

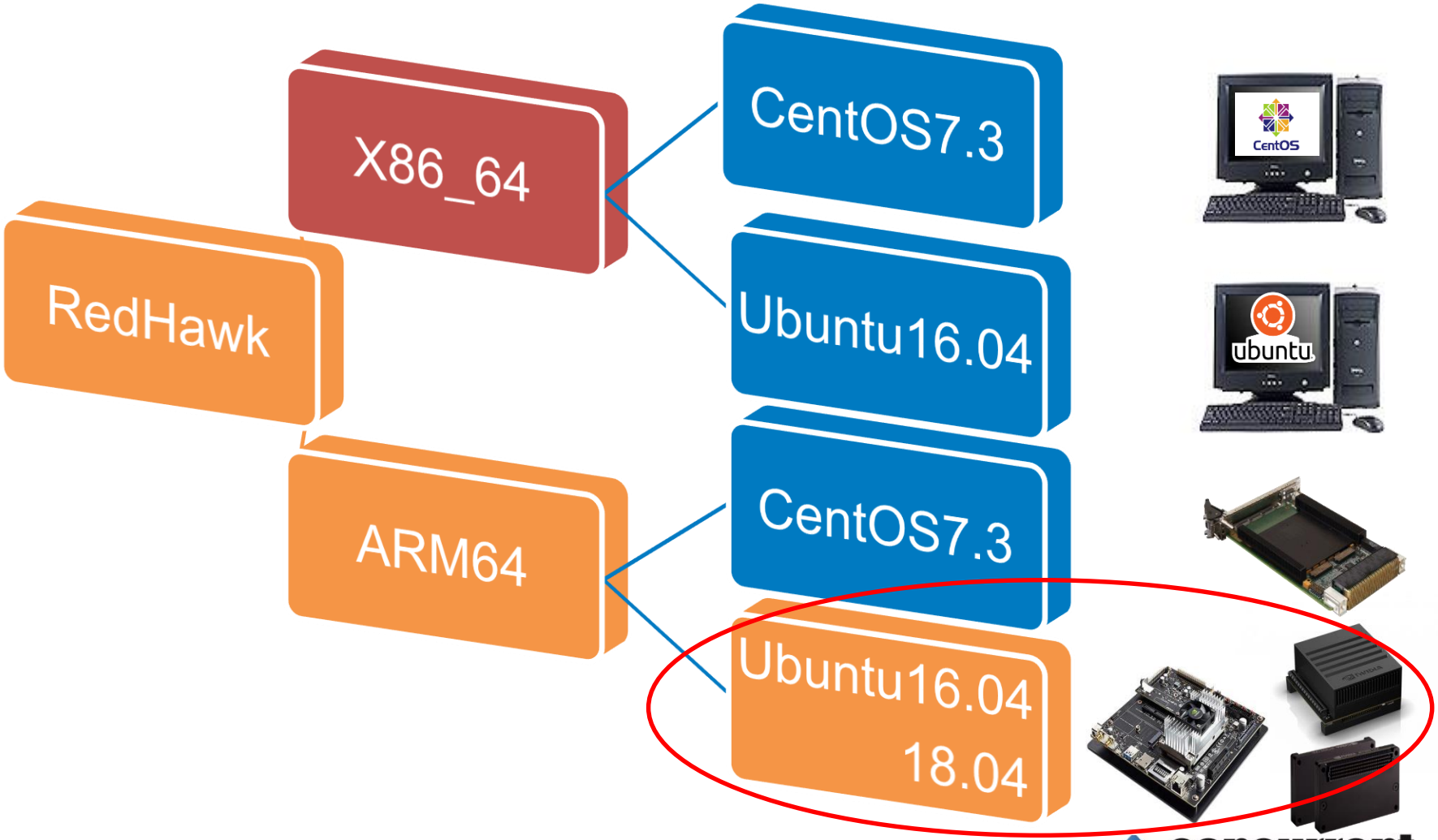
URL https://www.concurrent-rt.co.jp

mailto:info@concurrent-rt.co.jp

Kenrick Jackson CEO - Concurrent Real-Time Inc.

概略説明

8つのパッケージ


RedHawk
X86_64
ARM64
CentOS7.3
Ubuntu16.04
CentOS7.3
Ubuntu16.04
18.04
CentOS
ubuntu
concurrent
REAL-TIME

Jetson TX2 / Jetson AGX Xavier

	JETSON TX2	JETSON AGX XAVIER
GPU	256 Core Pascal @ 1.3GHz	512 Core Volta @ 1.37GHz 64 Tensor Cores
DL Accelerator	-	(2x) NVDLA
Vision Accelerator	-	(2x) 7-way VLIW Processor
CPU	6 core Denver and A57 @ 2GHz (2x) 2MB L2	8 core Carmel ARM CPU @ 2.26GHz (4x) 2MB L2 + 4MB L3
Memory	8GB 128 bit LPDDR4 58.4 GB/s	16GB 256-bit LPDDR4x @ 2133MHz 137 GB/s
Storage	32GB eMMC	32GB eMMC
Video Encode	(2x) 4K @30 HEVC	(4x) 4Kp60 / (8x) 4Kp30 HEVC
Video Decode	(2x) 4K @30 12 bit support	(2x) 8Kp30 / (6x) 4Kp60 12 bit support
Camera	12 lanes MIPI CSI-2 D-PHY 1.2 30Gbps	16 lanes MIPI CSI-2 | 8 lanes SLVS-EC D-PHY 40Gbps / C-PHY 109Gbps
PCI Express	5 lanes PCIe Gen 2 1x4 + 1x1 | 2x1 + 1x4	16 lanes PCIe Gen 4 1x8 + 1x4 + 1x2 + 2x1
Mechanical	50mm x 87mm 400 pin connector	100mm x 87mm 699 pin connector
Power	7.5W / 15W	10W / 15W / 30W

RedHawk for Jetsonの特徴

- 商用のハードリアルタイムLinux
  - **Linux For Tegra(L4T)は、ハードリアルタイムOSでは無い**
- **バイナリ互換(Jetpack のカーネルをリアルタイム化)**
  - ARM64ビットUbuntu 16.04/18.04
  - NVIDIA JetPack3.2.1/4.0 SDK
  - POSIX-API
  - ROS1/ROS2
- イージーインストール
- Concurrent Real Time社製 リアルタイムソフトウェア
  - ユーティリティプログラム & リアルタイム拡張ライブラリ
    - shield/run/cpu/libccur-rt/libfbs/e.t.c.
  - NightStar™ デバッギング&解析ツール
    - NightView™ ソースレベルデバッガ
    - NightTrace™ イベントアナライザ
    - NightProbe™ データモニタ
    - NightSim™　アプリケーションスケジューラ
    - NightTune™ パフォーマンスチューナ

RedHawk for Jetsonのラインアップ

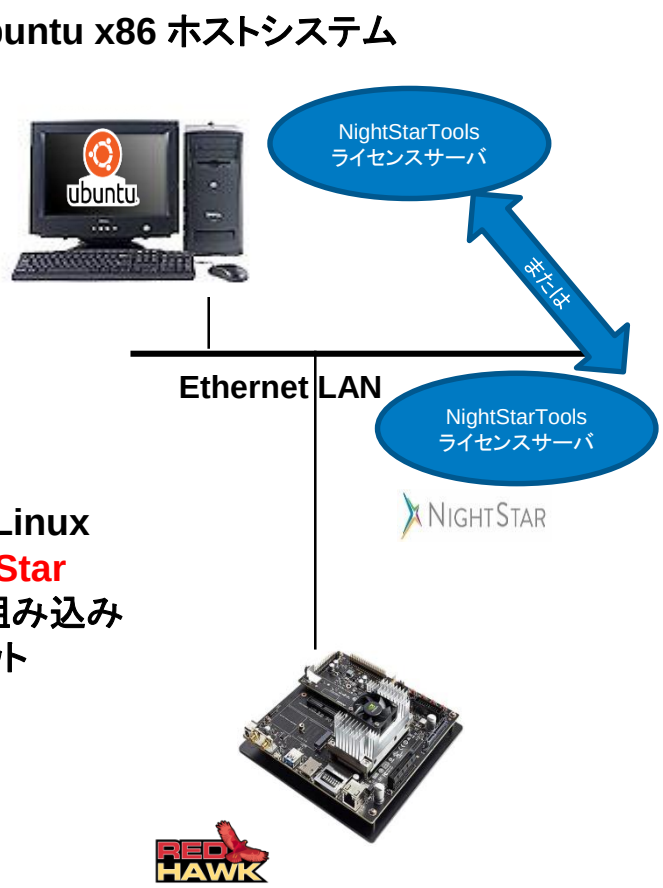
RedHawk	TX1	TX2	TX2i	Kernel	JetPack	NVIDIA L4T	Released
6.5	X			3.10.96	2.3	R24.2	October 2016
7.3		X		4.4.38-rt49	3.1	R28.1	November 2017
7.3.1		X	X	4.4.38-rt49	3.2	R28.2	May 2018
7.3.2	**X**	**X**	**X**	**4.4.38-rt49**	**3.2.1**	**TX1: R28.2 TX2: R28.2.1**	**July 2018**

RedHawk	AGX Xavier	Kernel	JetPack	NVIDIA L4T	Released
7.5	X	4.9.108	4.0	R31.0.1	November 2018

concurrent REAL-TIME

RedHawk for ARM64 Jetson開発環境

セルフ開発



- Jetpack 3.2.1 with L4T インストール
  - $ chmod +x JetPack-L4T-3.2.1-linux-x64_bxx.run
  - $ ./JetPack-L4T-3.2.1-linux-x64_bxx.run
- RedHawkインストーラのコピー
  - $ cd /media/ubuntu/RedHawk 7.3 aarch64/Packages
  - $ scp *.deb ubuntu@IPアドレス:/tmp
- RedHawkインストール(JetsonTx上で)
  - $ cd /tmp
  - $ sudo apt install ./ccur*.deb

NightStar™ デバッギング&解析ツール

- NightView™ ソースレベルデバッガ
- NightTrace™ イベントアナライザ
- NightProbe™ データモニタ
- NightSim™　アプリケーションスケジューラ
- NightTune™ パフォーマンスチューナ

NIGHTSTAR TOOLS

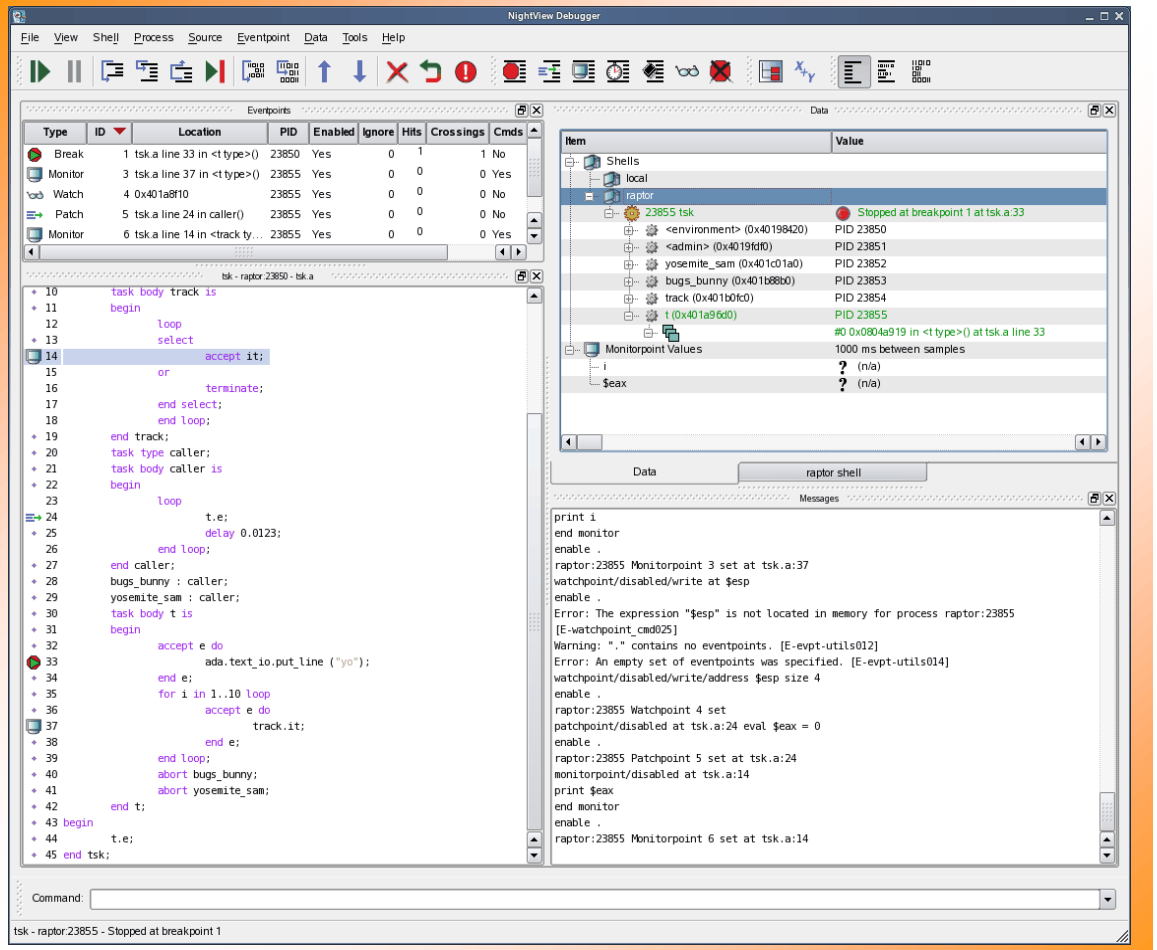
NightStar™ デバッギング&解析ツール概要

- 最小限のオーバーヘッドにてデバッグが可能
- アプリケーションのスピードで実行
- Linuxカーネルとユーザーアプリケーションのイベントを表示
- 全ての操作が画面上で完結
- 異なるCPU間で動作するアプリケーションのスレッドを表示
- C/C++, Fortran, Adaに対応
- x86, ARM64に対応

RedHawk NightView™ ソースレベルデバッガ

NightStar

- フレキシブルかつ堅牢なソースレベル・デバッガ
- 完全なアプリケーション速度で実行が可能
- マルチシステム, マルチプロセスを1つのGUIでデバッグが可能
- 動的メモリ領域のデバッグが可能

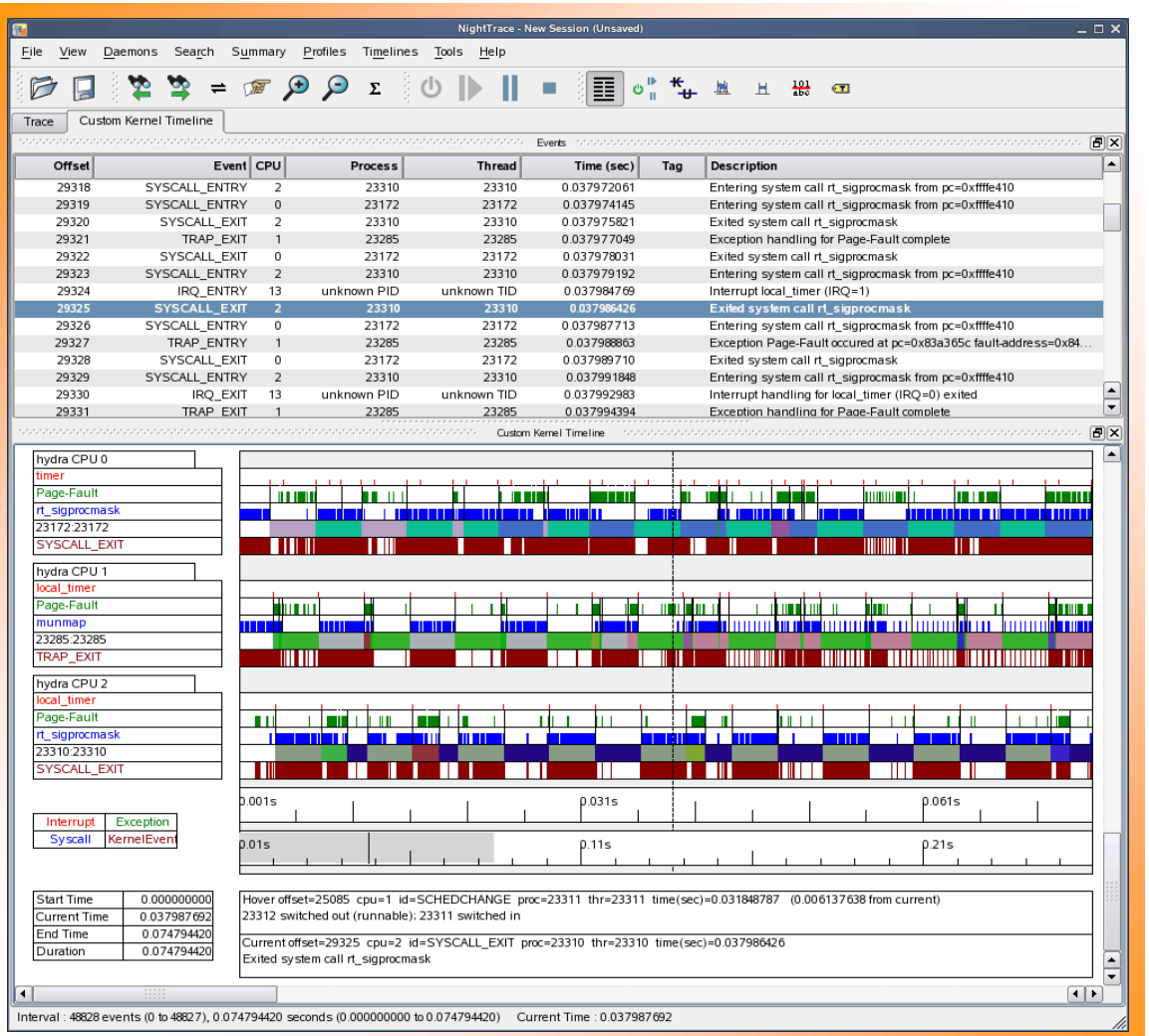


concurrent REAL-TIME

RedHawk NightTrace™ イベントアナライザ

NightStar

- フレキシブルかつ堅牢なイベント・アナライザー
- イベント, ステータス, データの表示を同期した時間ベースで提供
- アプリケーションの動作やカーネルに影響を正確に確認可能
- ユーザー定義イベントの記録が可能
- OSイベントはシステムコール, 割り込み, 例外処理を記録

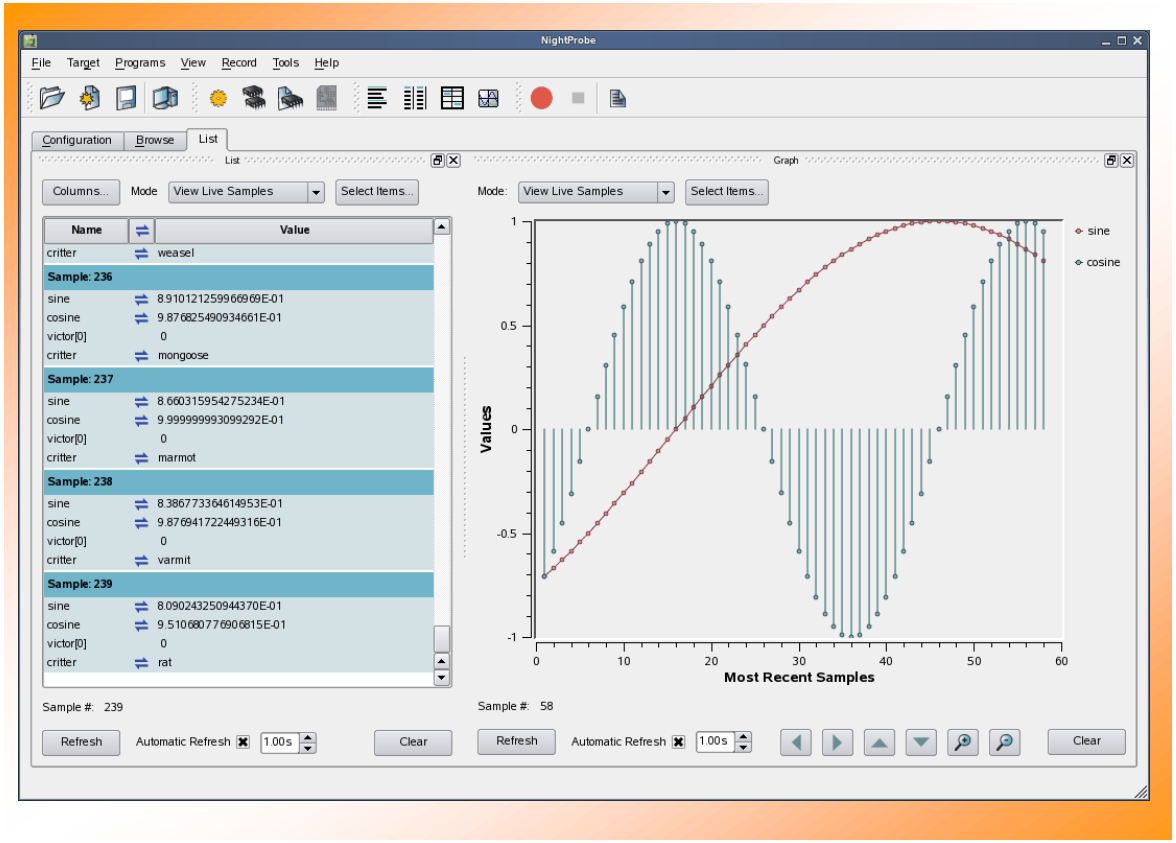


concurrent REAL-TIME

RedHawk NightProbe™ データモニタ

NightStar

- 動作への影響なくアプリケーションデータの監視, 表示, 記録が可能
- リアルタイムでのデータ表示、調査用にデータの記録が可能
- フレキシブルなデータ画面
- 実行中のプログラムのデータ修正が可能
- サンプリング、レコーディング用のAPIを提供

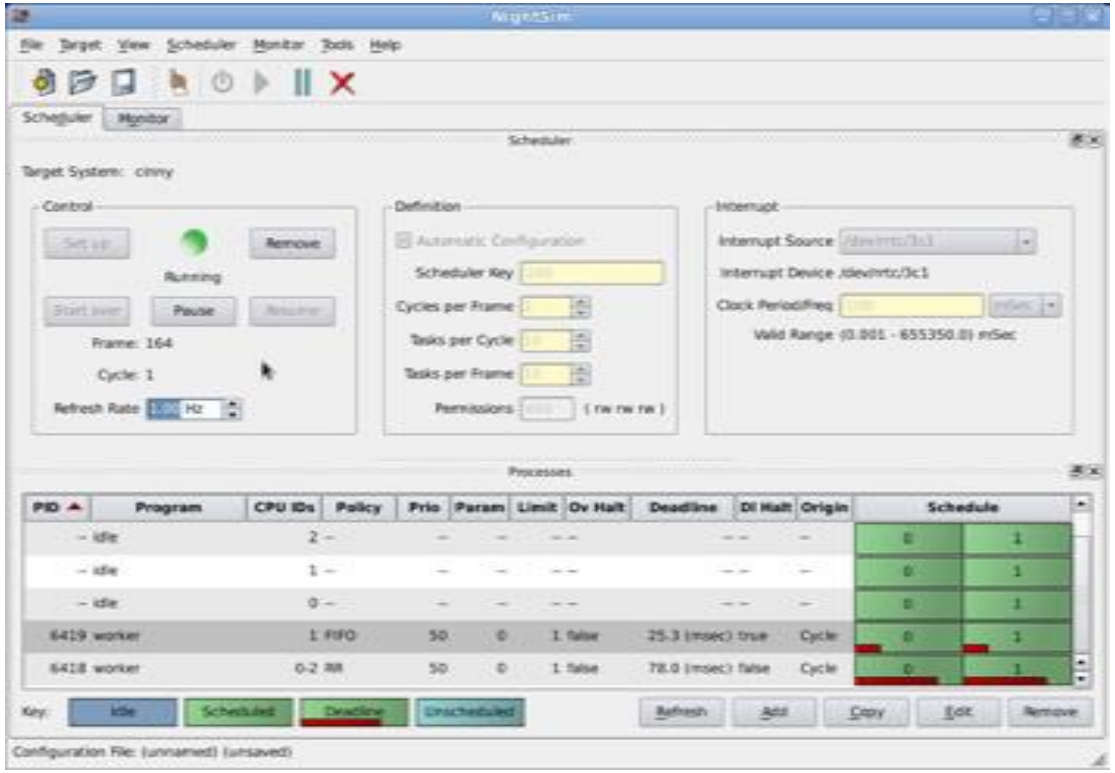


concurrent REAL-TIME

RedHawk NightSim™ アプリケーションスケジューラ

NightStar

- NightSimは、予測可能な周期的なプロセス実行を必要とするタイムクリティカルなアプリケーションのための定期的なスケジューラです。シミュレーション用途に最適なNightSimを使うと、開発者は複数の調整済みプロセス、優先順位、スケジューリングポリシー、CPU割り当ての実行を動的に調整できます。ユーザは、期間の実行を表示してアプリケーションのパフォーマンスを監視できます。

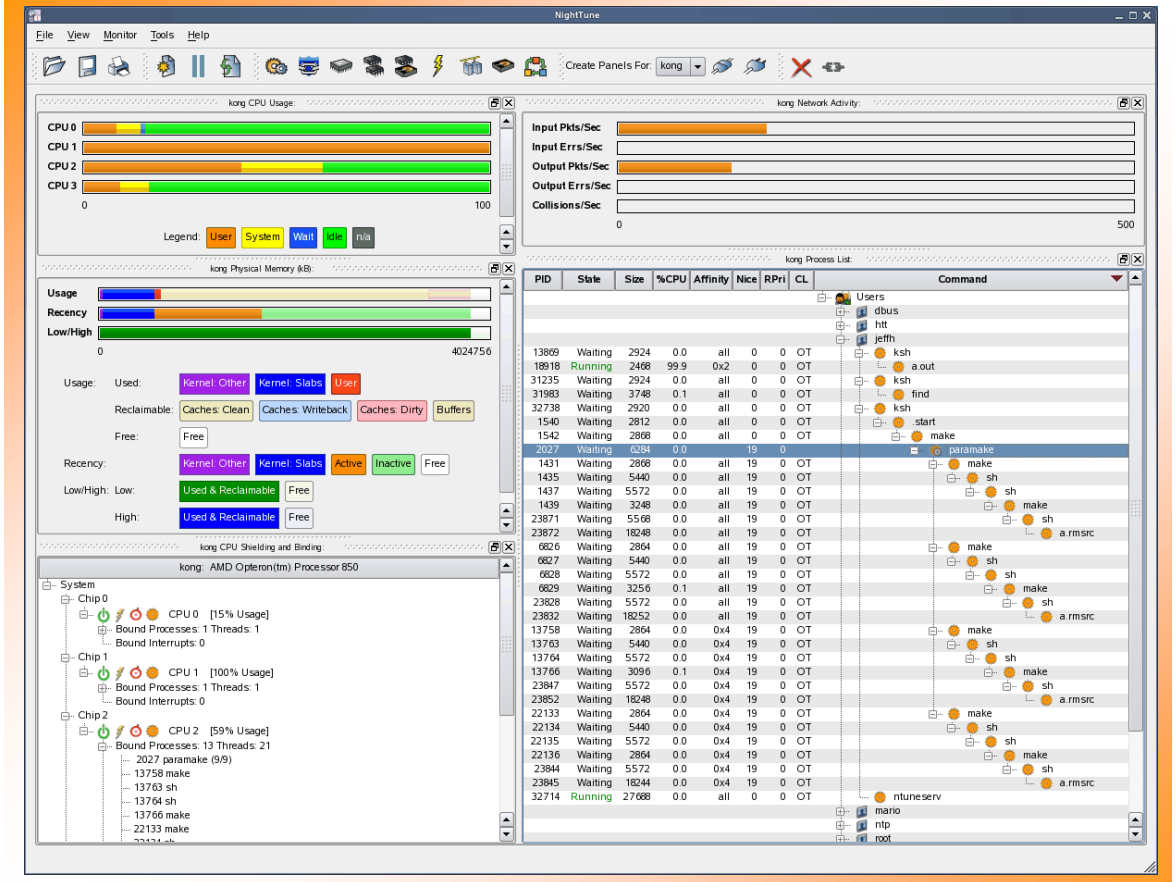


concurrent REAL-TIME

RedHawk NightTune™ パフォーマンスチューナ

NightStar

- チューニングを通してアプリケーションの性能を最適化
- システム状況の監視および制御が可能
- 優先度, スケジューリングポリシー, CPU割付け, 割込み, ネットワーク使用状況etcを監視
- GUIでCPU, 優先度, ポリシー, シールドetcを制御
- プロセス, スレッド, 割込みのCPUアフィニティを動的に制御



concurrent REAL-TIME

RedHawk for ARM64 Jetsonプロダクト情報

RedHawk for ARM64 Jetsonプロダクト情報

RedHawk Embedded for ARM64 Jetson TXn

- W-RHLE-DEVC-01-AJ

  RedHawk Embedded 1-user Development Package for ARM64 Jetson TXn

- W-RHLE-DEVC-02-AJ

  RedHawk Embedded 2-user Development Package for ARM64 Jetson TXn

- W-RHL-DEVC-01-A

  RedHawk Development Package Additional User Seat for ARM64 Platforms

- W-RHLE-TGT

  RedHawk Embedded Per-Target Charge

購入時1年間保証
標準サポート(SSS)
  Webサポートとリポジトリサービス
  ConcurrentProblemReport
ワンストップサポート
プロフェッショナルサポート

RedHawk for ARM64 Jetson保守サービス

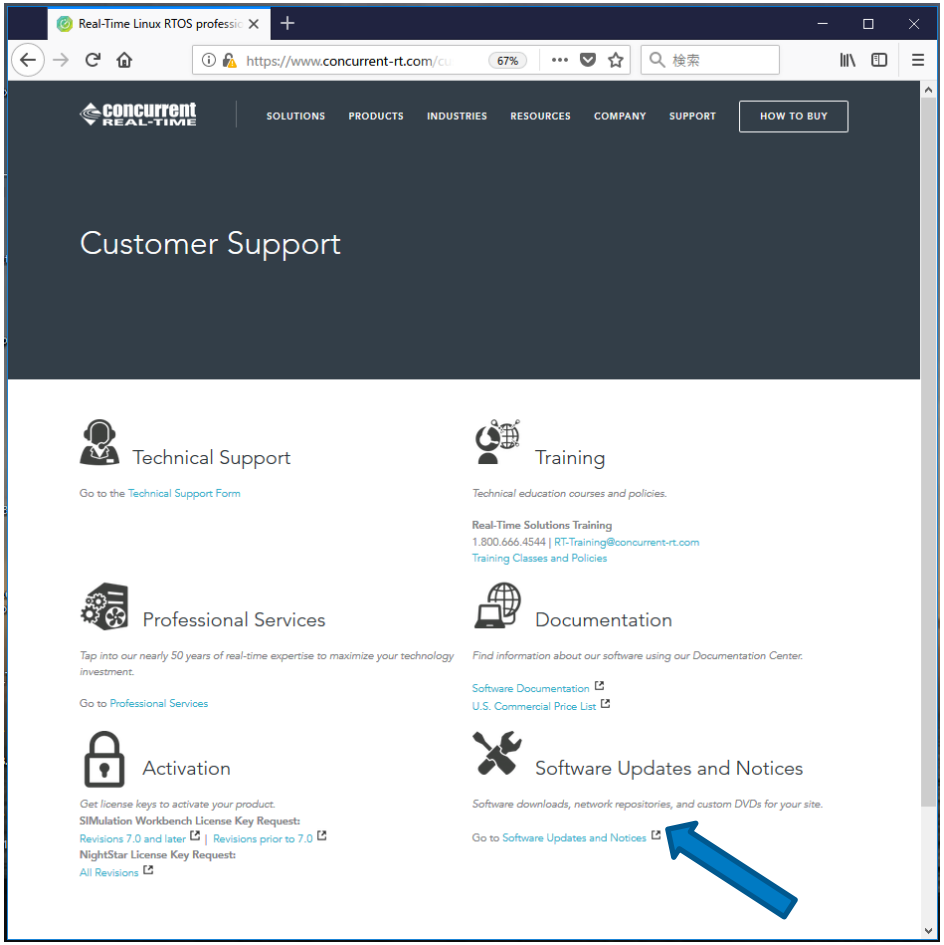
標準サポートサービス

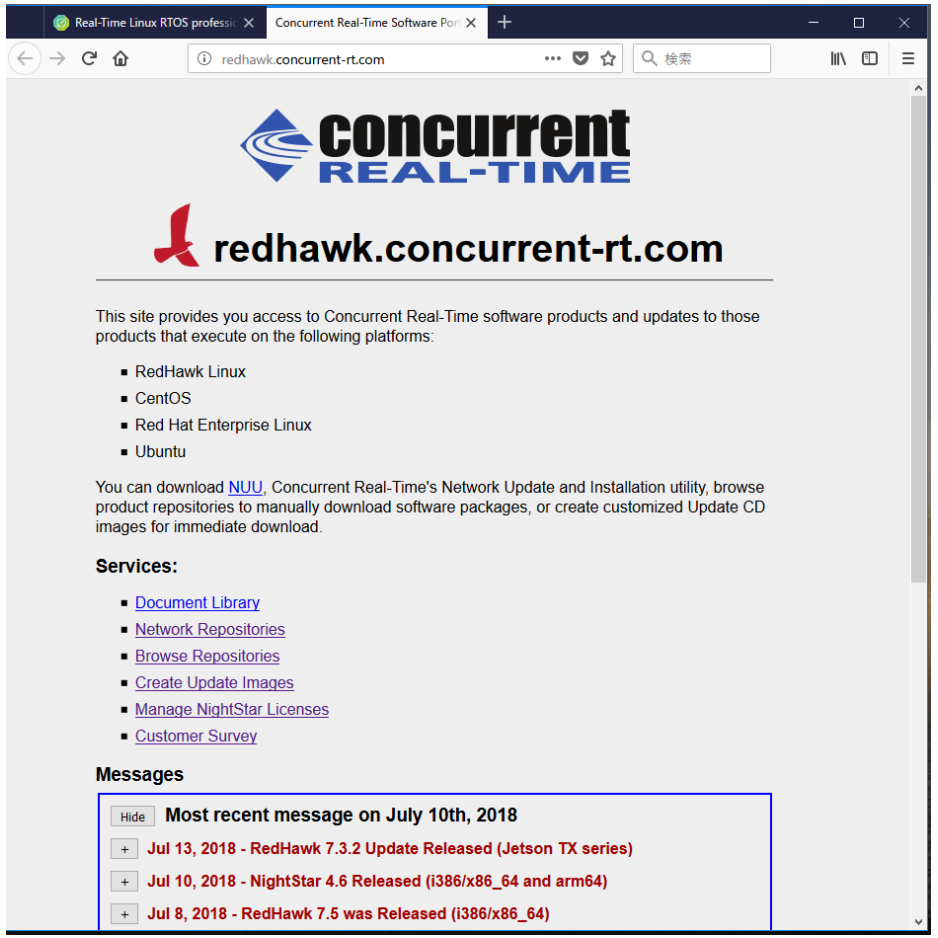
- RedHawk Linux for ARM64 Jetsonには、下記標準サポートサービスが含まれます。
  - 1年間の電子メール、Web、および電話によるテクニカルサポート

  （平日月曜～金曜日 9:00-17:00で、土日、祝祭日、年末年始、ゴールデンウィーク期間を除きます)。
  - RedHawk LinuxカーネルとNightStarツールのダウンロード可能なアップデートとパッチ。
  - RedHawkとNightStarの新リリース。
- RedHawk Linuxの標準サポートは、RedHawkのリアルタイムカーネル、システムコール、ライブラリ、およびConcurrent Real-Time社が開発したユーティリティを対象としています。

Webサポートとリポジトリサービス

https://www.concurrent-rt.com/customer-support/ 本社サイト
https://www.concurrent-rt.co.jp/customer-support/ 日本サイト



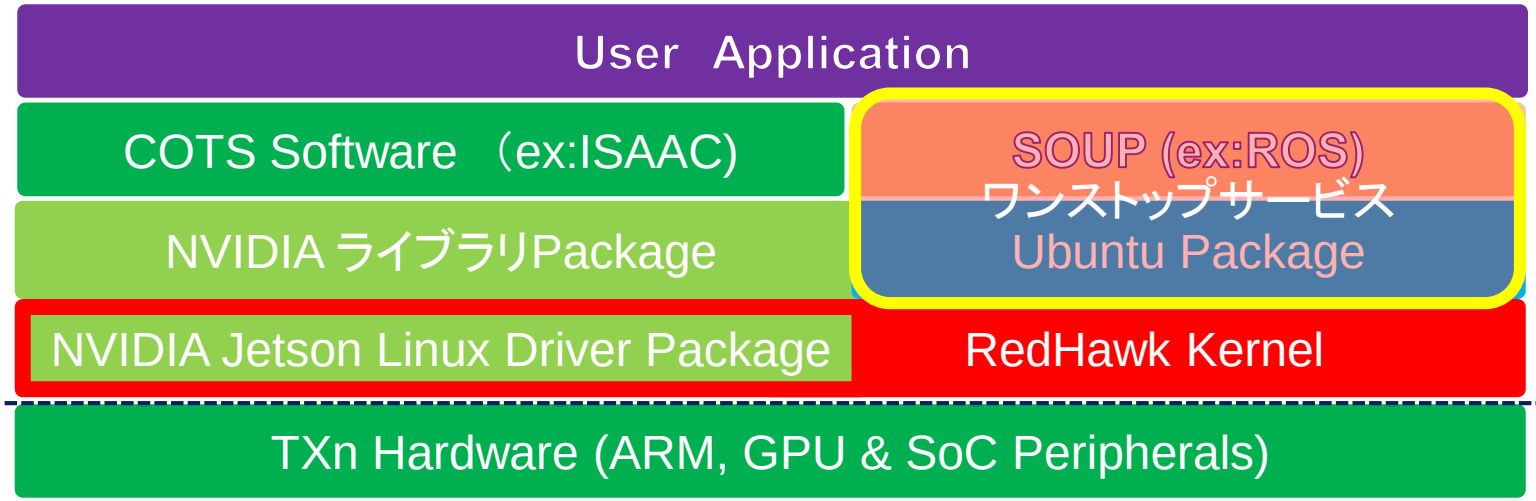


RedHawk ConcurrentProblemReport

- システム不具合解析情報収集コマンド
  - /etc/issue
  - /proc/cmdline
  - /proc/cpuinfo
  - /proc/interrupts
  - /proc/meminfo
  - /proc/modules
  - /proc/version
  - /proc/iomem
  - /var/log/syslog
  - /var/log/Xorg.0.log
  - /etc/X11/xorg.conf
  - /usr/bin/gather
  - /usr/bin/shield
  - /usr/bin/lspci
  - /proc/config.gz
  - /bin/dmesg
  - /sbin/lsirq -lm
  - /bin/netstat -i
  - /bin/netstat -s
  - /bin/netstat -g
  - /sbin/ifconfig
  - /sbin/ifconfig -a
  - /usr/sbin/arp -a
  - /usr/bin/dpkg -l
  - /bin/grep install /var/log/dpkg.log
  - /bin/systemctl list-unit-files

ワンストップサポートサービス

- RedHawk以外のソフトウェア、たとえばGNUソフトウェア、標準のLinuxディストリビューションライブラリ、Concurrent　Real-Time社によって変更または開発されていないコマンドおよびユーティリティのサポート等を、別途有償で提供します。



Ubuntu
長期サポート(LTS)版は、5年間に渡り**(Ubuntu16.04LTSは、2021/4まで)**セキュリティアップデートがコミュニティより無償で提供されます。(セキュリティアップデートのバックポートは、プロフェッショナルサポートで提供されます)

SOUP(Software of Unknown Provenance)
開発過程が不明なソフトウェア（SOUP）とは、正式な文書がない、または第三者によって開発され、かつ開発過程に対する管理について一切証拠がない、あらゆるコード（ツールまたはソースコード）

COTS（Commercial off-the-shelf）ソフトウェア
商用の3rdパーティのソフトウェア製品のサポートは、個別メーカとの契約に従って行います

RedHawk プロフェッショナルサポートサービス

- 下記のような要求に応じて、別途有償サポートを提供します。
  - デバイスドライバ開発
  - パフォーマンス分析とチューニング
  - アプリケーションの移植と開発
    - セキュリティアップデートのバックポート
  - リアルタイムシステムの適合評価
  - FTA (Fault Tree Analysis)
  - ベンチマークテスト
  - 各種トレーニング
  - コンサルティング、その他

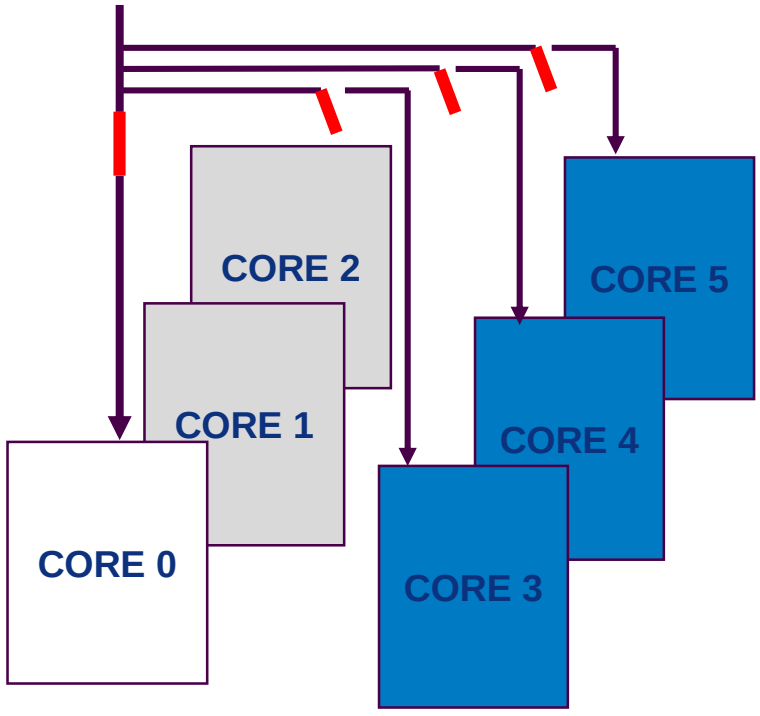
RedHawk for ARM64 Jetson

リアルタイム性能評価

shield & cpu コマンド

- CPUコアをシールド(割り込みをブロック)するコマンド

  ⇒L4Tでは、sudo echo 1 > /proc/irq/default_smp_affinity が近い。



Unshielded CORE

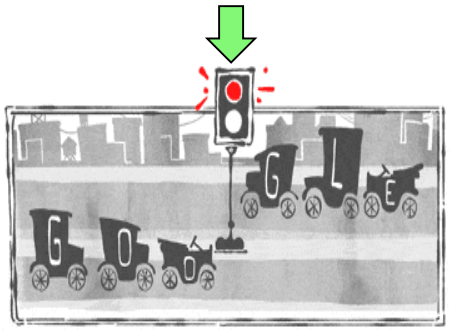
Shielded CORE

InterruptとCORE0のキャッシュ変動等の影響を受けない

Down CORE

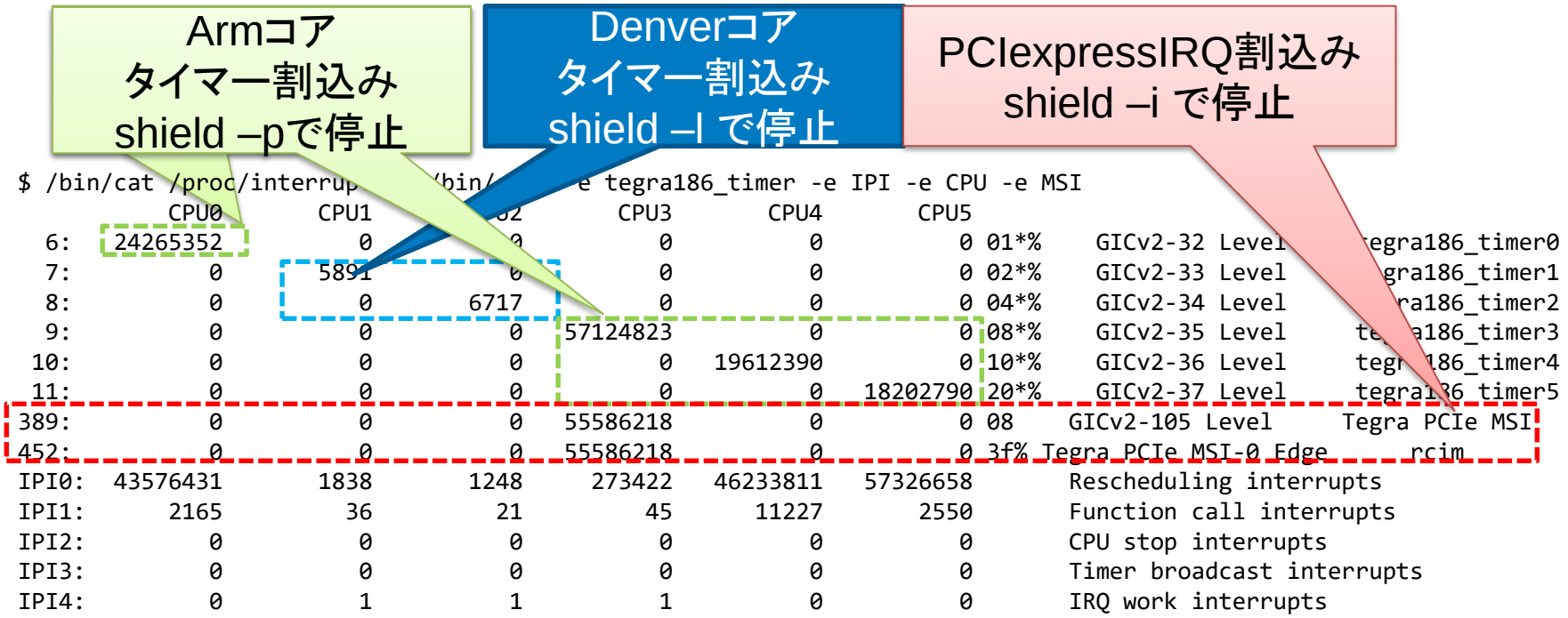
使用できなくする(スケジュールしない)

割り込み



concurrent REAL-TIME

shieldのsmp_affinity制御




```
$ /bin/cat /proc/interrup      /bin/      e tegra186_timer -e IPI -e CPU -e MSI
           CPU0       CPU1       CPU2       CPU3       CPU4       CPU5
  6:   24265352          0          0          0          0          0 01*%    GICv2-32 Level      tegra186_timer0
  7:          0       5891          0          0          0          0 02*%    GICv2-33 Level      gra186_timer1
  8:          0          0       6717          0          0          0 04*%    GICv2-34 Level      ra186_timer2
  9:          0          0          0   57124823          0          0 08*%    GICv2-35 Level      te  a186_timer3
 10:          0          0          0          0   19612390          0 10*%    GICv2-36 Level      tegr 186_timer4
 11:          0          0          0          0          0   18202790 20*%    GICv2-37 Level      tegra186_timer5
389:          0          0          0   55586218          0          0 08    GICv2-105 Level     Tegra PCIe MSI
452:          0          0          0   55586218          0          0 3f% Tegra PCIe MSI-0 Edge      rcim
IPI0:  43576431       1838       1248     273422   46233811   57326658       Rescheduling interrupts
IPI1:      2165         36         21         45      11227       2550       Function call interrupts
IPI2:         0          0          0          0          0          0       CPU stop interrupts
IPI3:         0          0          0          0          0          0       Timer broadcast interrupts
IPI4:         0          1          1          1          0          0       IRQ work interrupts
```

- IRQ389　Tegra PCIe MSIがTegra PCIe MSI-0を制御しているので、PCIeの割込みベクトルはこのsmp_affinityを変更する
- sudo echo 8 > /proc/irq/389/smp_affinity

CPUコマンドは動的にCPUコアをUP/DOWN

```
$ cpu
 cpu chip core   ht  ht-siblings  state  shielding
 --- ---- ----  ---  -----------  -----  ---------
   0    1    0    -  -            up     none
   1    0    0    -  -            down   none
   2    0    1    -  -            down   none
   3    1    1    -  -            up     proc irq ltmr
   4    1    2    -  -            up     none
   5    1    3    -  -            up     none

$ cpu -C
cpu    shared_cpus    level      size    type          n-way   line_sz
---    -----------    -----      ----    ----          -----   -------
  0    -                  1       32K    Data              2        64
  0    -                  1       48K    Instruction       3        64
  0    0,3-5              2     2048K    Unified          16        64

  1    -                  1        0K    Data              1        64
  1    -                  1        0K    Instruction       1        64
  1    1-2                2        0K    Unified           1        64

  2    -                  1        0K    Data              1        64
  2    -                  1        0K    Instruction       1        64
  2    1-2                2        0K    Unified           1        64

  3    -                  1       32K    Data              2        64
  3    -                  1       48K    Instruction       3        64
  3    0,3-5              2     2048K    Unified          16        64

  4    -                  1       32K    Data              2        64
  4    -                  1       48K    Instruction       3        64
  4    0,3-5              2     2048K    Unified          16        64

  5    -                  1       32K    Data              2        64
  5    -                  1       48K    Instruction       3        64
  5    0,3-5              2     2048K    Unified          16        64
```

Denverコアには、スケジュールされない

キャッシュを共有しているCPUグループ

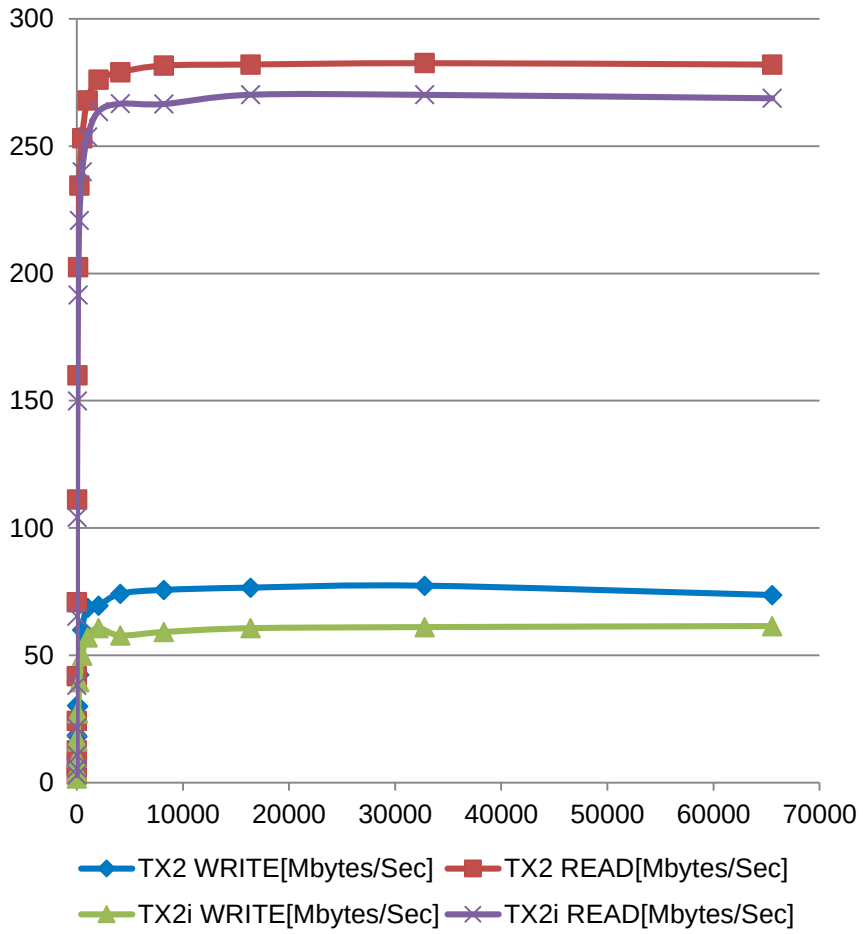
Denverコアはキャッシュを持っていない

POSIX DIRECT DISK TRANSFER API

Tx2/Tx2i FlashDisk速度

Tx2iのFlash Disk　性能は10%減

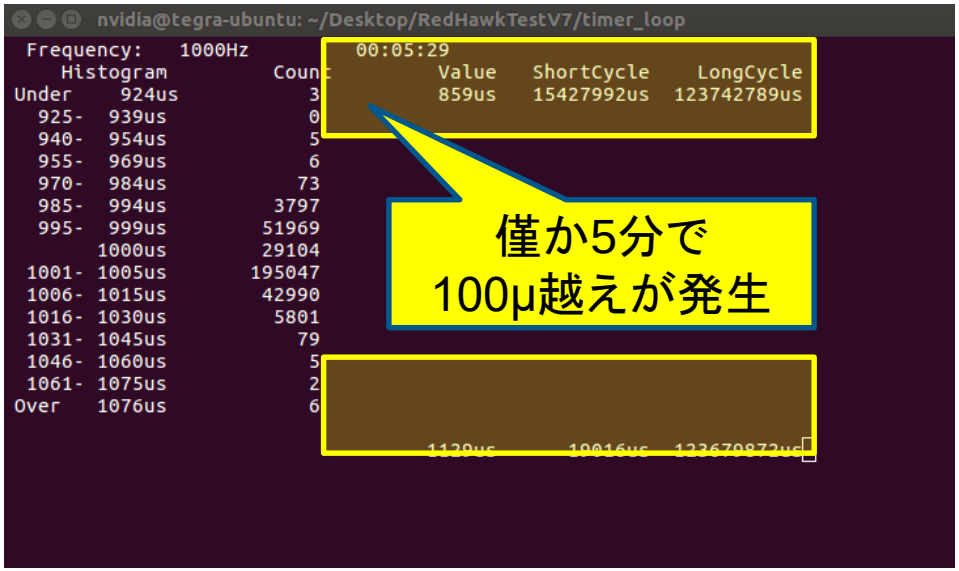
Buffer Size KiBytes	TX2 WRITE[Mbytes/Sec]	TX2 READ[Mbytes/Sec]	TX2i WRITE[Mbytes/Sec]	TX2i READ[Mbytes/Sec]
0.5	1.77	3.43	1.67	3.01
1	3.48	6.52	3.14	5.74
2	4.54	12.74	4.26	10.85
4	5.77	24.33	5.47	21.57
8	10.86	41.87	9.8	38.28
16	18.61	70.88	17.55	65.41
32	30.26	111.2	27.42	104.16
64	18.08	159.99	16.2	149.94
128	30	202.47	27.34	191.48
256	42.38	234.54	39.46	220.74
512	59.94	253.03	49.92	239.96
1024	68.66	267.98	57.13	253.61
2048	69.53	276.05	60.69	263.56
4096	74.18	279.02	57.83	266.61
8192	75.7	281.6	59.16	266.5
16384	76.59	282.04	60.67	270.14
32768	77.37	282.56	61.1	270.12
65536	73.71	282	61.52	268.75



concurrent REAL-TIME

POSIX TIMER API

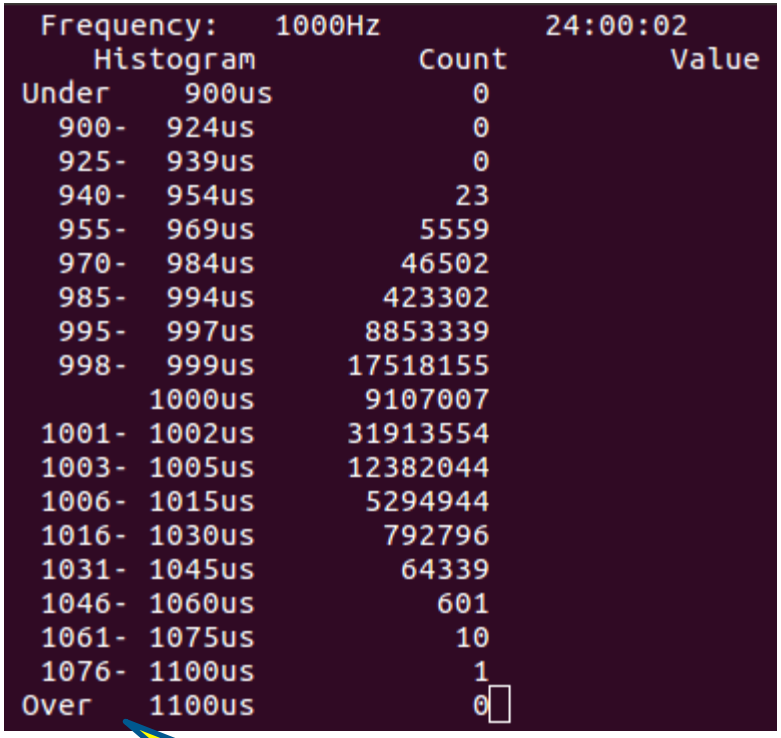
NVIDIAカーネル



CORE	SHIELDED	Programs
0	No	Interrupt,Daemon stress --cpu1 --io 1 --vm 1 --hdd 1
1	No	DOWN
2	No	DOWN
3	No	stress --cpu1 --io 1 --vm 1 --hdd 1
4	No	stress --cpu1 --io 1 --vm 1 --hdd 1
5	Yes	ベンチマークテストプログラム

ベンチマーク以外の
CPUには、100%負荷

RedHawkカーネル



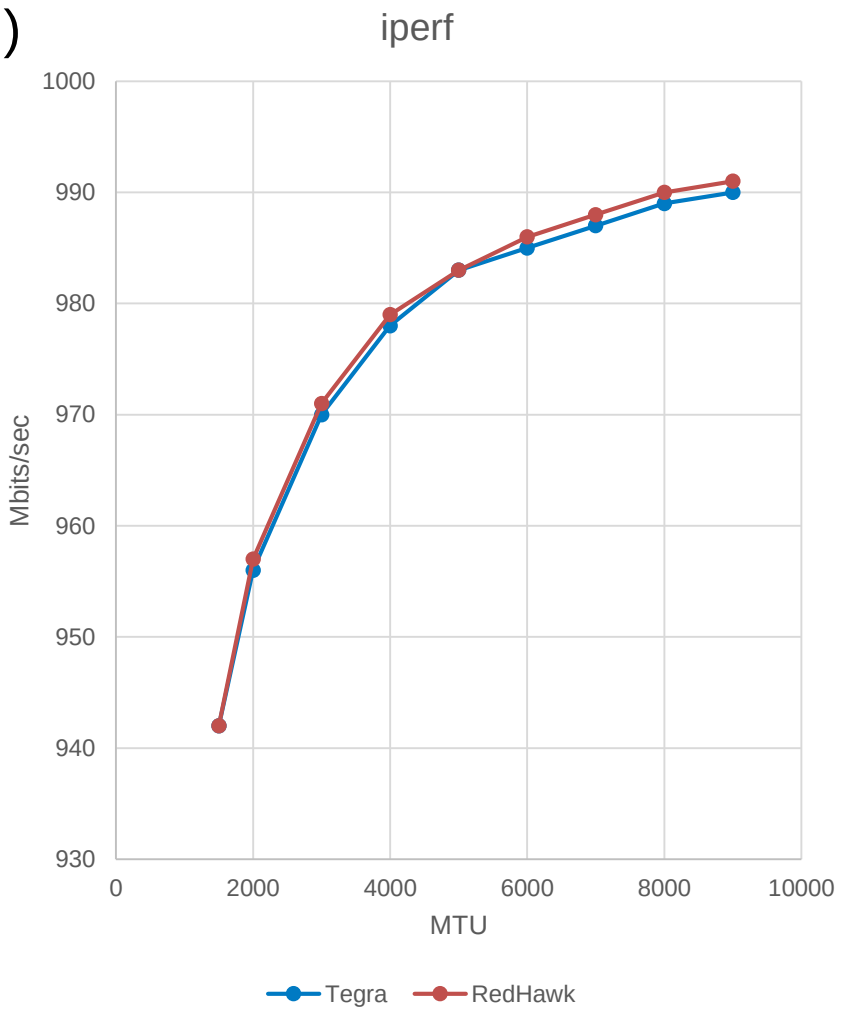
24H試験でも
100マイクロ秒以下

concurrent REAL-TIME

オンボードETHERNET性能(iperf)

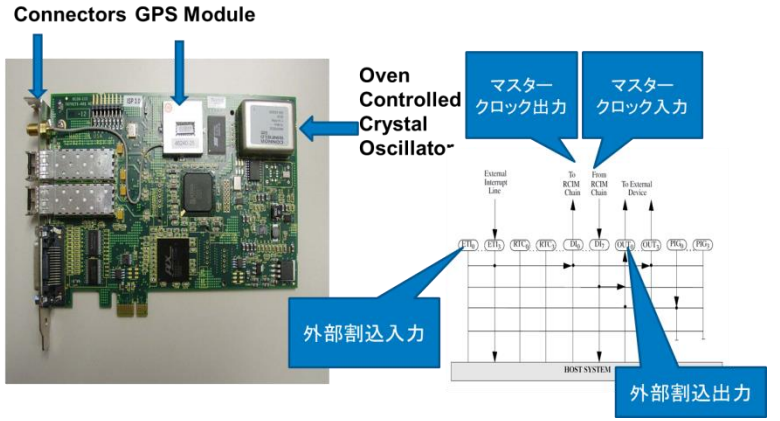
- RedHawkが高速(プリエンプション効果)

MTU	NVIDIA(Tegra) Mbits/sec	RedHawk Mbits/sec
1500	942	942
2000	956	957
3000	970	971
4000	978	979
5000	983	983
6000	985	986
7000	987	988
8000	989	990
9000	990	991



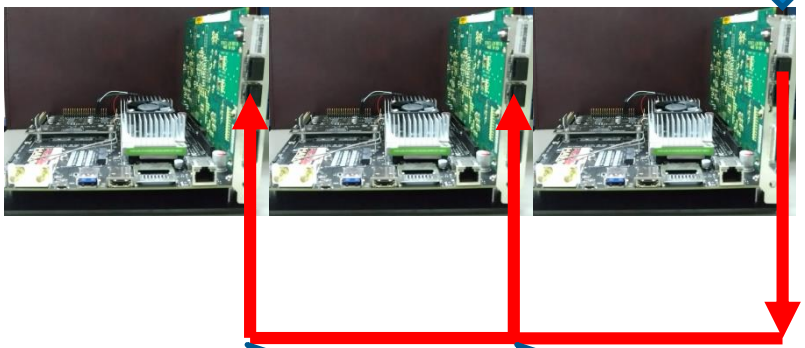
Real-Time Clock & Interrupt Module

- RedHawk for ARM64 Jetsonには、外部イベントへの迅速な対応が必要な、タイムクリティカルなアプリケーション向けに設計された多機能PCIeカードであるReal-Time Clock&Interrupt Module (RCIM) をオプションで搭載することができます。
- RCIMには、複数のJetsonシステムで読み取り可能な同期クロック、8つのプログラマブルタイマ、12の入力および12出力の外部割り込みラインが含まれています。RCIMは、RedHawk Linuxによって完全にサポートされています。オプションのオンボードGPSモジュールを使用して、RCIMの同期クロックをGPS標準時間に合わせることができます。1つのGPS搭載RCIMは、RCIMチェーン内のすべてのJetsonを同期できます。
- また、GPSモジュールを搭載した複数のJetsonは、システム間のケーブル接続なしに、共通のタイムベースから動作できます。GPS絶対時間に基づくPOSIXタイマは、物理的に接続されていないシステム上でプログラムの実行を同時に開始するために使用できます。



GPS アンテナ

RCIM:精度±2.5PPM



800ns遅れ　400ns遅れ

RCIMによる時刻調整(1)

RCIMなし

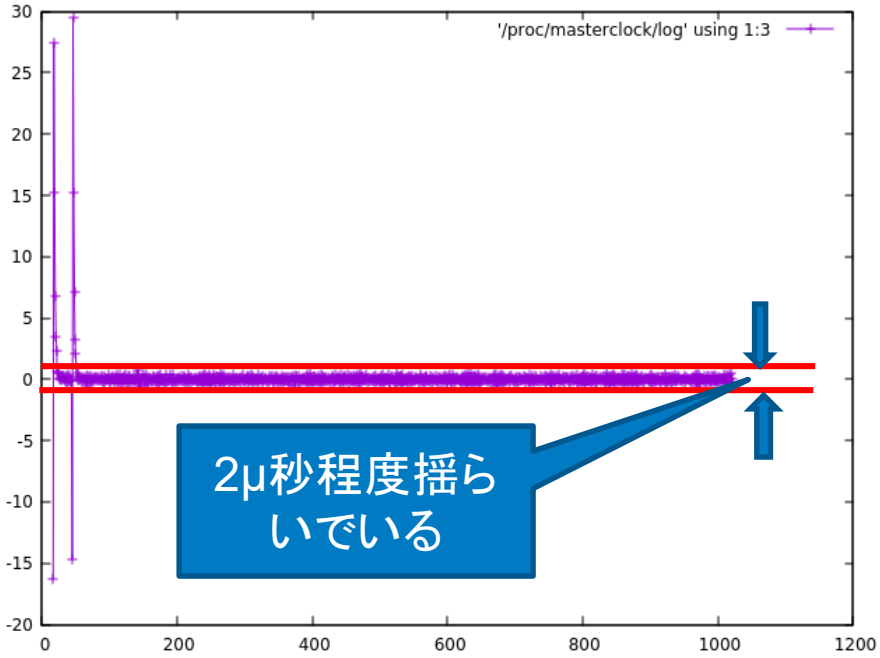
```
$ cat /proc/masterclock/available
none
$ cat /proc/masterclock/current
none
$ cat /proc/masterclock/status
              masterclock: none
        masterclock resolution: 0 nsecs
              system clock: arch_sys_counter
        sync status: disabled, this is a NOP masterclock
            adjtimex freq adj: 0.000 usecs/s
      adjtimex phase adj rate: 0.000 usecs/s


$ cat /proc/masterclock/log
# Masterclock status log.
# col 1: timestamp (CLOCK_MONOTONIC)
# col 2: total phase to be corrected for (usecs)
# col 3: currently applied phase correction (usecs/s)
# col 4: currently applied frequency correction (usecs/s)
# col 5: desired change in frequency correction (usecs/s)
# col 6: adjtimex frequency adjustment (usecs/s)
# col 7: adjtimex phase adjustment (usecs/s)
# col 8: adjtimex tick size (usecs/tick)
# col 9: clocksource multiplier, active
# col 10: clocksource multiplier, raw
# col 11: clocksource timestamp, delta
```

RCIMあり

```
$ cat /proc/masterclock/available
rcim:0 (rcim) none
$ cat /proc/masterclock/current
rcim:0 (rcim)
$ cat /proc/masterclock/status
                    masterclock: rcim:0 (rcim)
        masterclock resolution: 400 nsecs
                  system clock: arch_sys_counter
                   sync status: running
            syncing interval: 999.999 msecs (1.000 Hz)
                 instability: 15 (very stable)
                           phase: -0.324 usecs
                  phase adjustment: -0.324 usecs/s
       delta frequency adjustment: -0.162 usecs/s
             frequency adjustment: -2.381 usecs/s
                      fetch time: 2.944 usecs
                  delta fetch time: 0.000 usecs
                       selected hz: auto
                phase & freq shifts: 0 & 1
                  adjtimex freq adj: 0.000 usecs/s
           adjtimex phase adj rate: 0.000 usecs/s
            --- RCIM-II/III masterclock internals ---
                           #skips: 0
                            #adds: 0
                        #tasklets: 0
```

RCIMによる時刻調整(2)

```
$ cat /proc/masterclock/log
# Masterclock status log.
# col 1: timestamp (CLOCK_MONOTONIC)
# col 2: total phase to be corrected for (usecs)
# col 3: currently applied phase correction (usecs/s)
# col 4: currently applied frequency correction (usecs/s)
# col 5: desired change in frequency correction (usecs/s)
# col 6: adjtimex frequency adjustment (usecs/s)
# col 7: adjtimex phase adjustment (usecs/s)
# col 8: adjtimex tick size (usecs/tick)
# col 9: clocksource multiplier, active
# col 10: clocksource multiplier, raw
# col 11: clocksource timestamp, delta
```

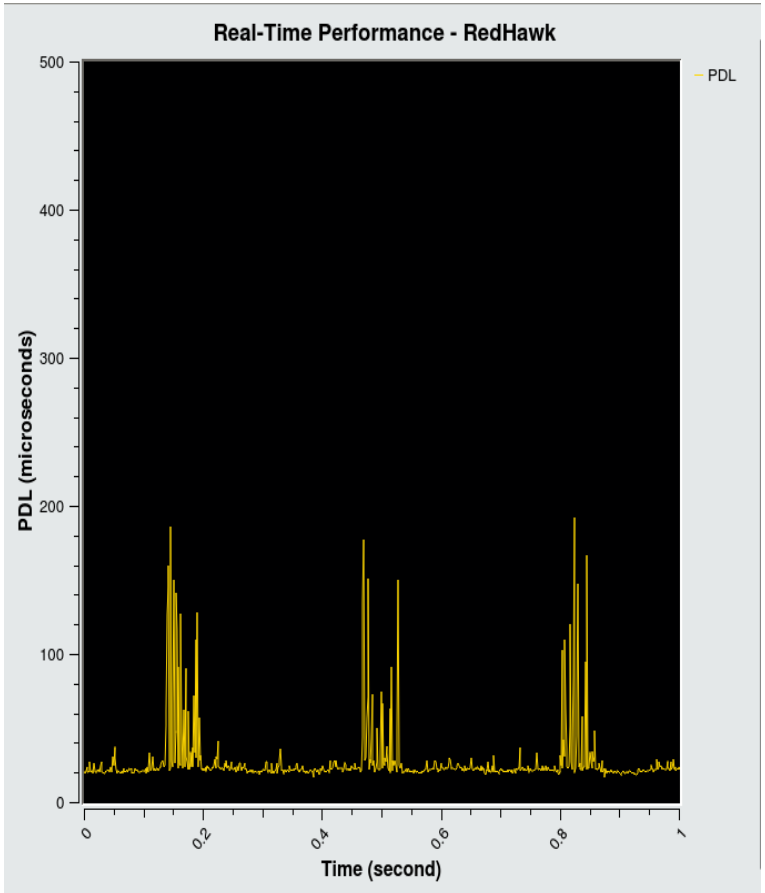


Cyclictest & Stress

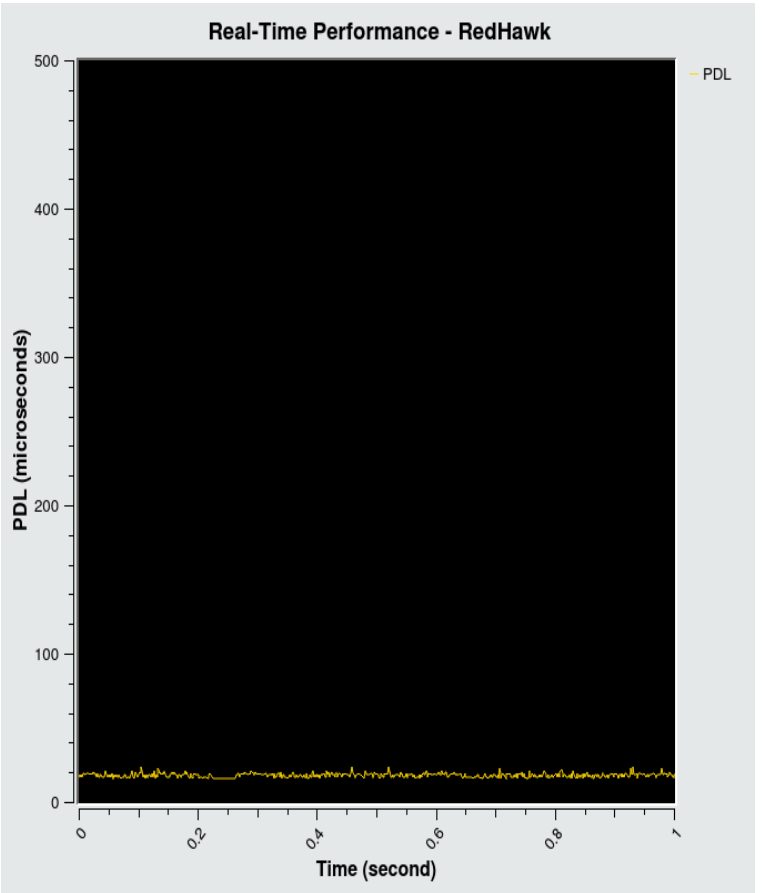
- Cyclictest & Stress : 最もポピュラーな評価プログラム
  - CPU上でCyclictestを実行します。
    https://rt.wiki.kernel.org/index.php/Cyclictest
  - シンプルなPOSIX Stressユーティリティを使用してバックグラウンドの負荷を与えます。
    http://people.seas.harvard.edu/~apw/stress/
  - 結果を連続的にサンプリングしグラフ化します

Cyclictest & Stress 結果

非シールド状態



シールド状態

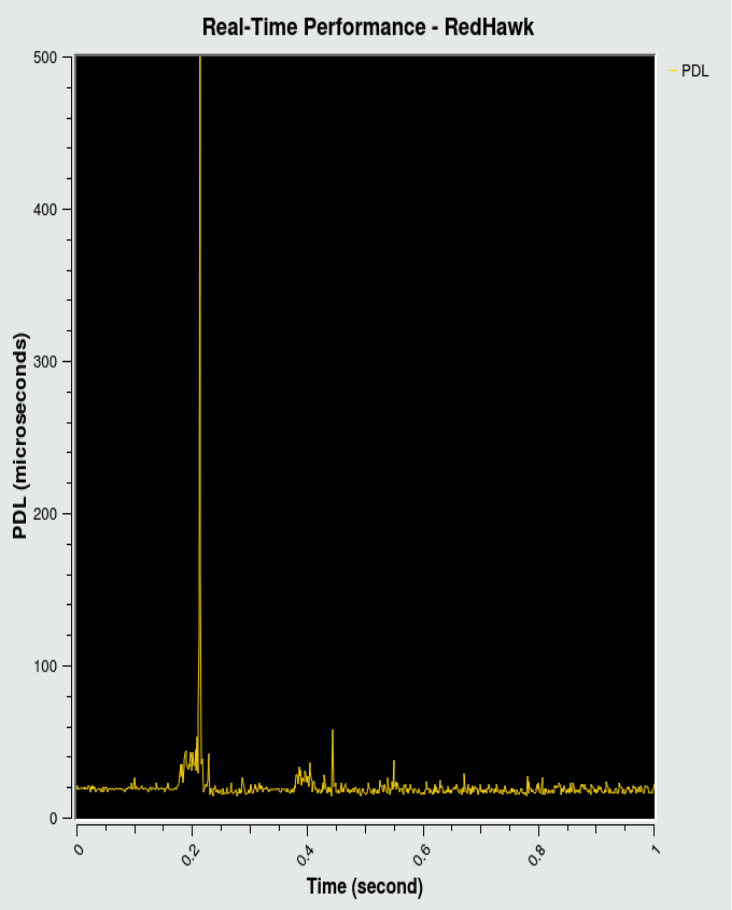


CUDA Reduction & Stress

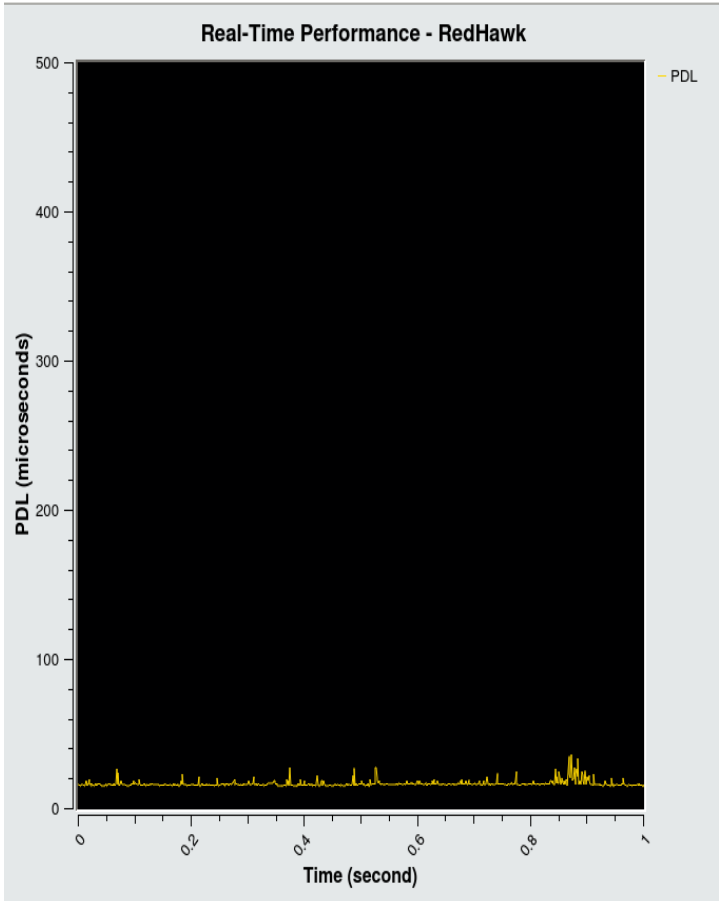
- CUDAアクティビティの影響の測定
  - リアルタイムパフォーマンスに対するCUDAアクティビティの影響の測定
  - CPU上でCyclictestリアルタイムデモを実行します。
  - CUDA Reductionのサンプルプログラムを使用してバックグラウンドの負荷を作成します。
    - 注　reductionは、配列の総和の計算で
      並列性の高いCUDAサンプルプログラム(GPU利用率が高い)
  - 結果を連続的にサンプリングしグラフ化します

CUDAアクティビティの影響の測定

- 非シールド状態



- シールド状態



concurrent REAL-TIME

12時間連続試験結果

- NVIDIA kernelの場合
  - カーネルでの自発的なプリエンプションの不足のためにcyclictestを実行できません
  - このベンチマークでは、最悪の場合ジッター時間は5ミリ秒を超えます

- RedHawk kernelの場合

RedHawk with Stress load on Jetson

最小6us 平均8us 最大38us

RedHawk with CUDA load on Jetson

最小6us 平均8us 最大48us

Using ROS with RedHawk™ on Jetson TXn

ROS1 / ROS2 ユースケース

ROS1

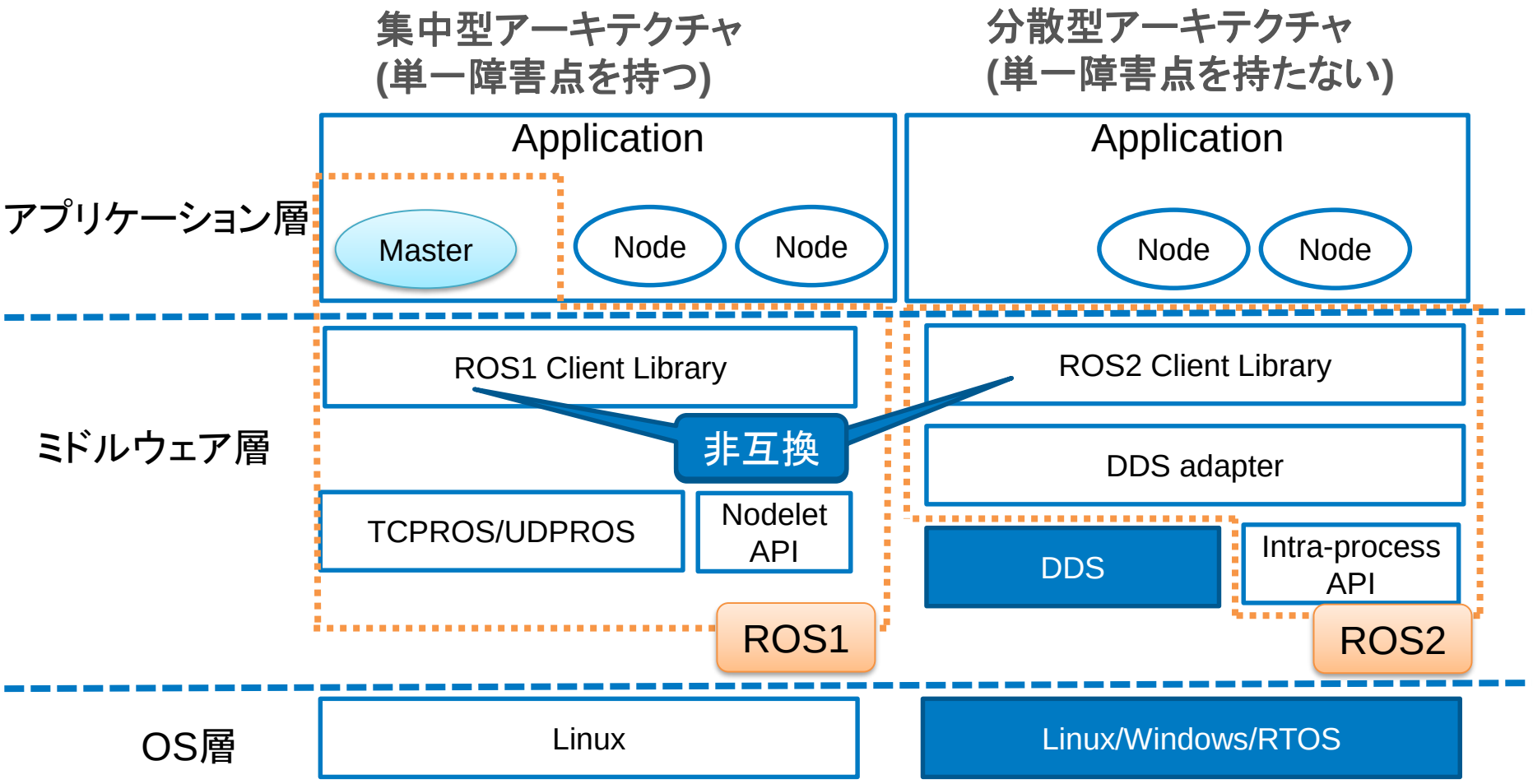
- 単一のロボット
- ワークステーションクラスのコンピュータ
- リアルタイム制御は求めない
  - ros_controlの特別な仕組みで別途対応
- 優れたネットワーク接続性
  - 有線または近接の高帯域幅ワイヤレス
- 主に学問分野の研究応用

ROS2

- 複数ロボットのチーム
- 小型の組み込みプラットフォーム
- リアルタイムシステム
  - プロセス間やマシン間の通信を含む、ROSで直接リアルタイム制御をサポート
- 非理想的なネットワーク
  - 低品質のWiFiや地上間通信リンクで、損失や遅延のためにネットワーク接続が劣化した場合でも、ROSを動作させる
- プロダクション環境
  - ROSが研究室で選択され続けるプラットフォームであり続けることは不可欠だが、ROSベースのラボプロトタイプが現実のアプリケーションでの使用に適したROSベースの製品に進化できる。
- プロジェクト範囲の縮小
  - 積極的に外部のオープンソースやCOTSプロダクトを取り入れる

⇒ROS2では、制御と通信のリアルタイム性が重要

ROS1 / ROS2 設計思想



ROS 1.0

- ROS1のインストール

```
$ sh -c 'echo "deb http://packages.ros.org/ros/ubuntu $(lsb_release -sc) main" > /etc/apt/sources.list.d/ros-latest.list‘
$ apt-key adv --keyserver hkp://ha.pool.sks-keyservers.net:80 --recv-key 421C365BD9FF1F717815A3895523BAEEB01FA116
$ apt-get update
$ apt-get install ros-kinetic-desktop-full
$ rosdep init
```

- ROSを搭載したNvidia Jetson TK1、TX1、またはTX2でCSIカメラを簡単に使用できるROSパッケージ。
  - https://github.com/peter-moran/jetson_csi_cam
- GStreamerベースのビデオストリーム用のROSカメラドライバ。
  - https://github.com/ros-drivers/gscam
  - https://github.com/ros-drivers/gscam/issues/3

ROS 2.0

- ROSはRTOSではありませんが…
  - ROS 2.0は、リアルタイムに対応し、基礎となるRTOSと緊密に協力するように設計されています
- ROS 2.0はRedHawk OSを非常に効果的に使用できます
- 下記、URLのホワイトペーパーにROS2インストールを含む詳細を記述しています。
  - “Using ROS with RedHawk Linux on the NVIDIA Jetson TX2”
  - ROS 2.0には、倒立振子のサンプルデモが含まれています
  - TX2上の最大PDLを15ミリ秒から50マイクロ秒まで改善しました

https://www.concurrent-rt.com/wp-content/uploads/2016/09/Using-ROS-with-RedHawk-on-Jetson-TX2.pdf

https://github.com/dusty-nv/

@NVIDIA Jetson開発者　ダスティン・フランクリン

- **jetson-inference**
  - TensorRTとNVIDIA Jetsonを使って深層学習推論ネットワークと深視覚プリミティブを導入するガイド
- **jetson-reinforcement**
  - PyTorch、OpenAI Gym、およびGazeboロボティクスシミュレータを搭載したNVIDIA JetsonのGPUライブラリ。
- **jetson-scripts**
  - NVIDIA Jetsonプラットフォームの設定スクリプトとインストールガイド
- **jetson-utils**
  - NVIDIA Jetson TX1 / TX2用のC / C ++ Linuxユーティリティラッパー - カメラ、コーデック、CUDA、GStreamer、HID、OpenGL / XGL

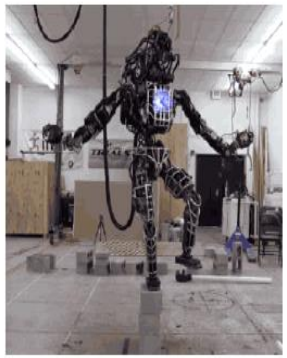
Real-time control in ROS and ROS 2.0

- **Jackie Kay**

Real-time control in ROS and ROS 2.0

Jackie Kay
jackie@osrfoundation.org

Adolfo Rodriguez Tsouroukdissian
adolfo.rodriguez@pal-robotics.com



PAL ROBOTICS

Open Source Robotics Foundation

ROS 2 Real-time Benchmarking: Results

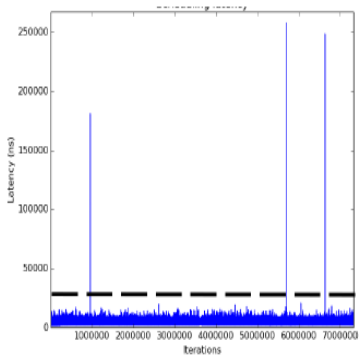
Stress applied:

```
stress --cpu 2 --io 2
```

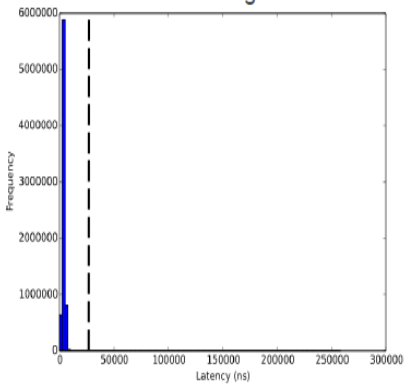
7,345,125 cycles observed

	Latency (ns)	% of update rate
Min	1398	0.14%
Max	258064	25.8%
Mean	3729.11	0.38%

3 instances of overrun observed



Jitter histogram



51

Goal

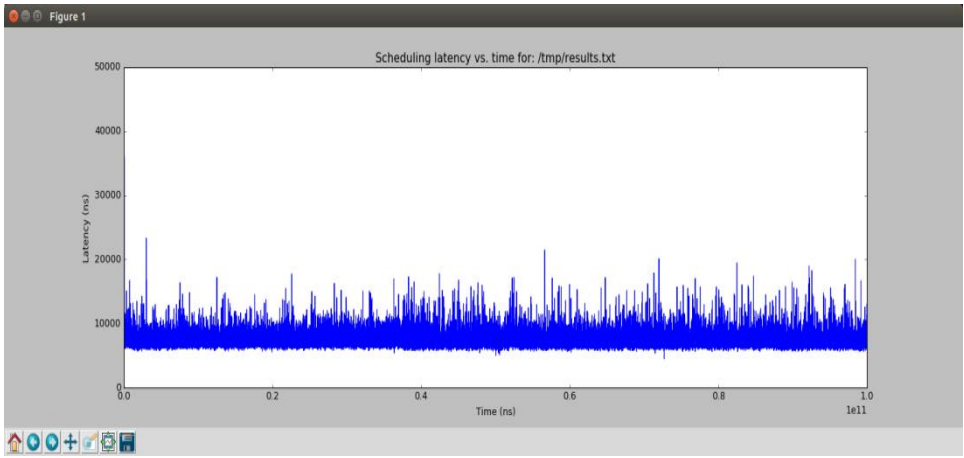
- 1 kHz update loop (1 ms period)
- Less than 3% jitter (30 µs)

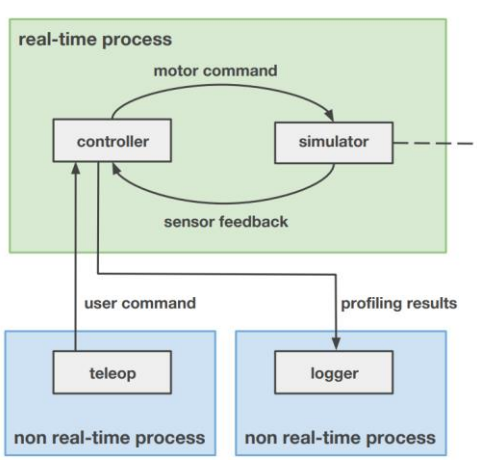
concurrent REAL-TIME

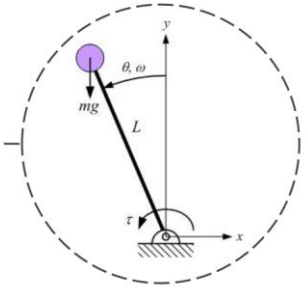
Using ROS2 with RedHawk on Jetson

- 倒立振り子のデモ（stress付き）

- 指令モータ角度：1.570796
- 実際のモータ角度：1.570404
- 現在の待ち時間：7005 ns
- 平均待ち時間：8021.6ns
- 最小待ち時間：6555 ns
- 最大待ち時間：47488 ns
- 実行中のマイナーページフォールト：0
- 実行中のメジャーページフォールト：0







pendulum_control

- **Latency (ns) % of update rate**

- **RT_PREEMPT kernel**
- **Round robin scheduler (SCHED_RR),**
- **thread priority: 98**

- Min 1398
- Max 258064
- Mean 3729.11

- **RedHawk RealTime Kernel(L4T)**
- **First in First out scheduler (SCHED_FIFO),**
- **thread priority: 90**

- Min 6555
- Max 47488
- Mean 8021.6

```
CPUID irqs ltmrs procs
------------------------
--
 0    no    no    no
 1    no    no    no
 2    no    no    no
 3    no    no    no
 4    no    no    no
 5    yes   yes    yes
```

最大値に注目

シールド

TX2は、平均速度からCPUが遅い事が読み取れる
しかし、PDLの最大値は47μ秒で5倍安定している。

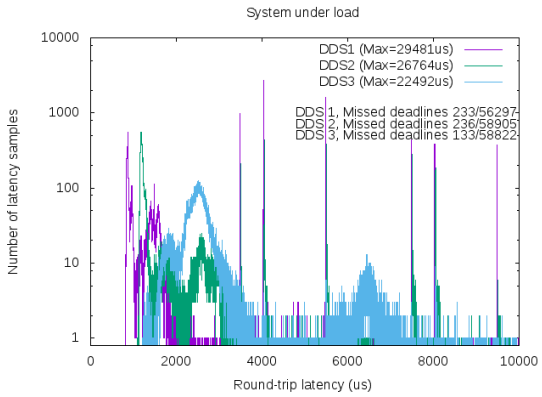
concurrent REAL-TIME

リアルタイムロボットアプリケーションのためのROS 2.0通信評価

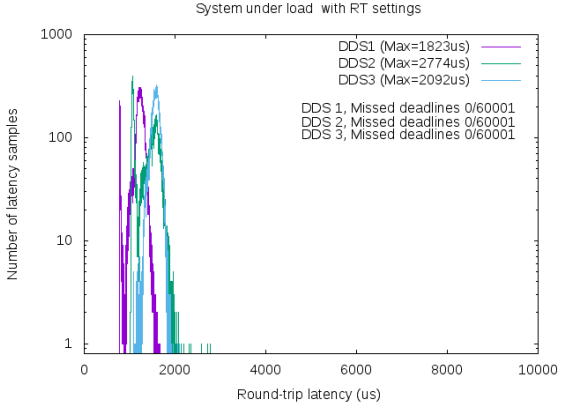
Carlos San Vicente Gutiérrez, Lander Usategui San Juan,
Irati Zamalloa Ugarte, Víctor Mayoral Vilches著

- DDSの実装では、LNS（Linux Network Stack）がトランスポート層とネットワーク層として使用されます。
- これにより、LNSはROS 2.0のパフォーマンスにとって重要な部分になります。
- しかし、ネットワークスタックは、限られたレイテンシのために最適化されるのではなく、与えられた瞬間のスループットのために最適化されます。
- イーサネット割り込みスレッドの優先度を他のIRQスレッドよりも高く設定して、ネットワークの決定性を向上させることができます。
- このアプローチは、通常の状態では、合理的な決定論的な振る舞いを期待できますが、ネットワークとシステムが高負荷の場合、(高優先度ノードの)待ち時間が大幅に増加する可能性があります。
- 現時点での、リアルタイムアプリケーション用にROS 2.0通信を最適化する最善の戦略は、
  a）リアルタイム設定でアプリケーションを構成する、
  b）アプリケーション設定に応じてカーネルスレッドを構成する、
  c）通信に関与する各マシンのネットワークとCPUの使用を制限する、
  事です。

高付加時の通信応答(ノーマル設定)

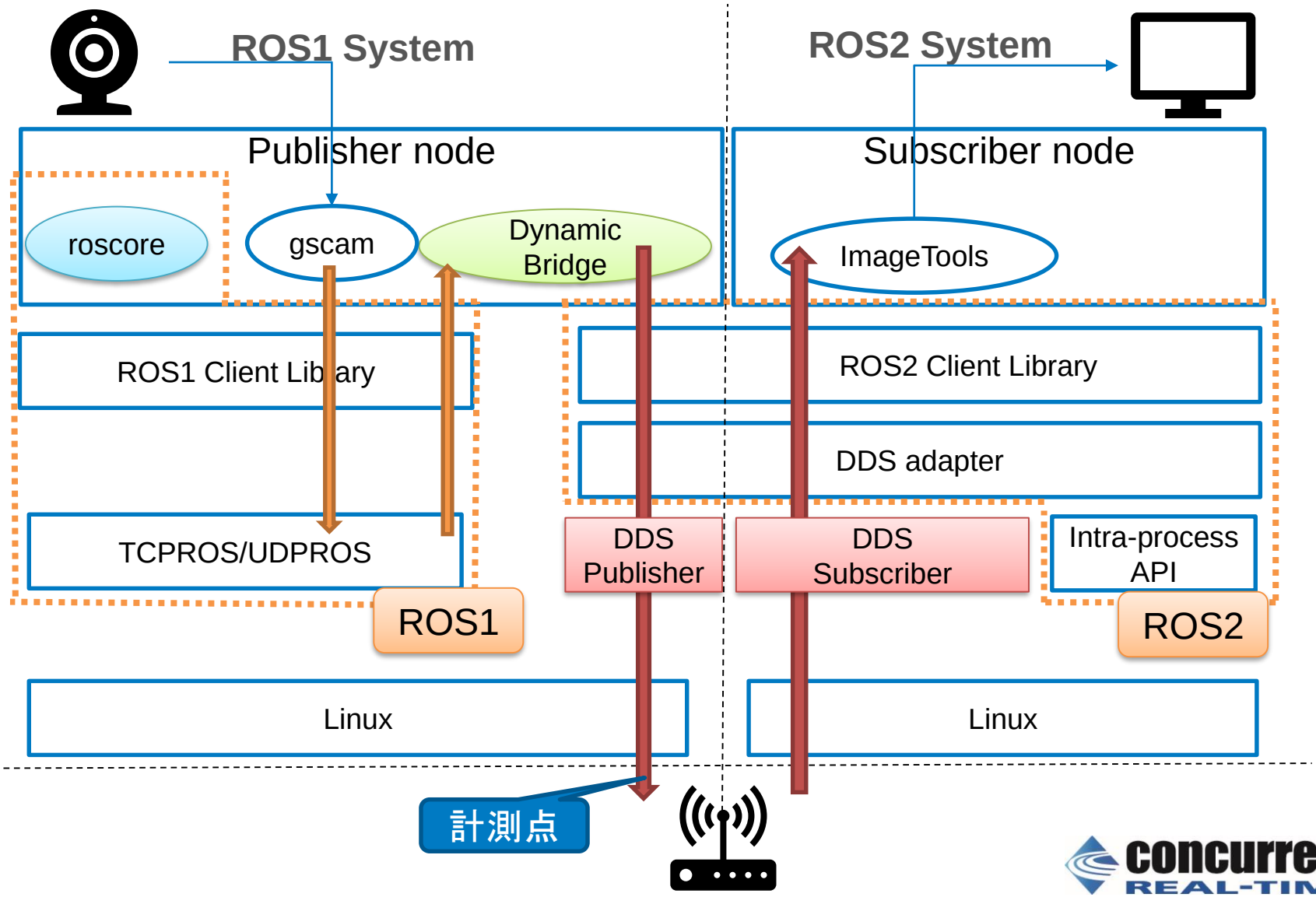


高付加時の通信応答(リアルタイム設定)



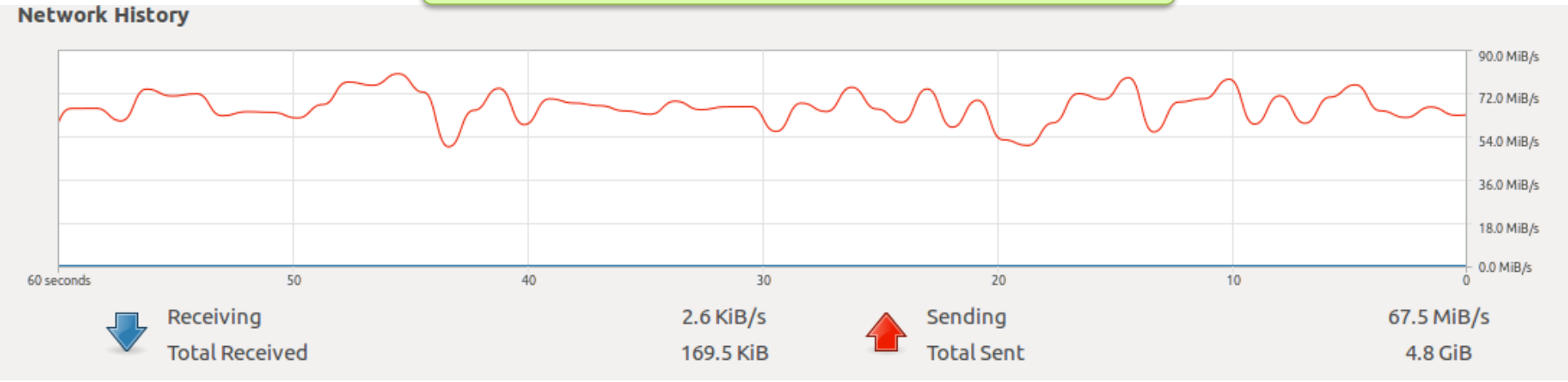
PDF版 https://arxiv.org/pdf/1809.02595.pdf
Web 版 https://hackernoon.com/towards-a-distributed-and-real-time-framework-for-robots-469ba77d6c42

ROS1 / ROS2 ブリッジ

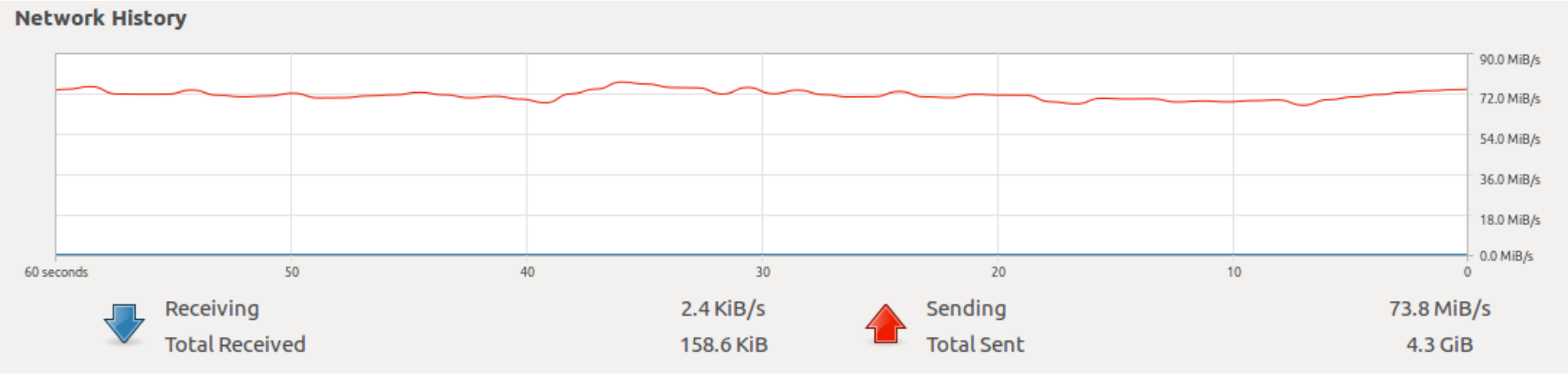


ROS2 DDS Publisher/Subscriber

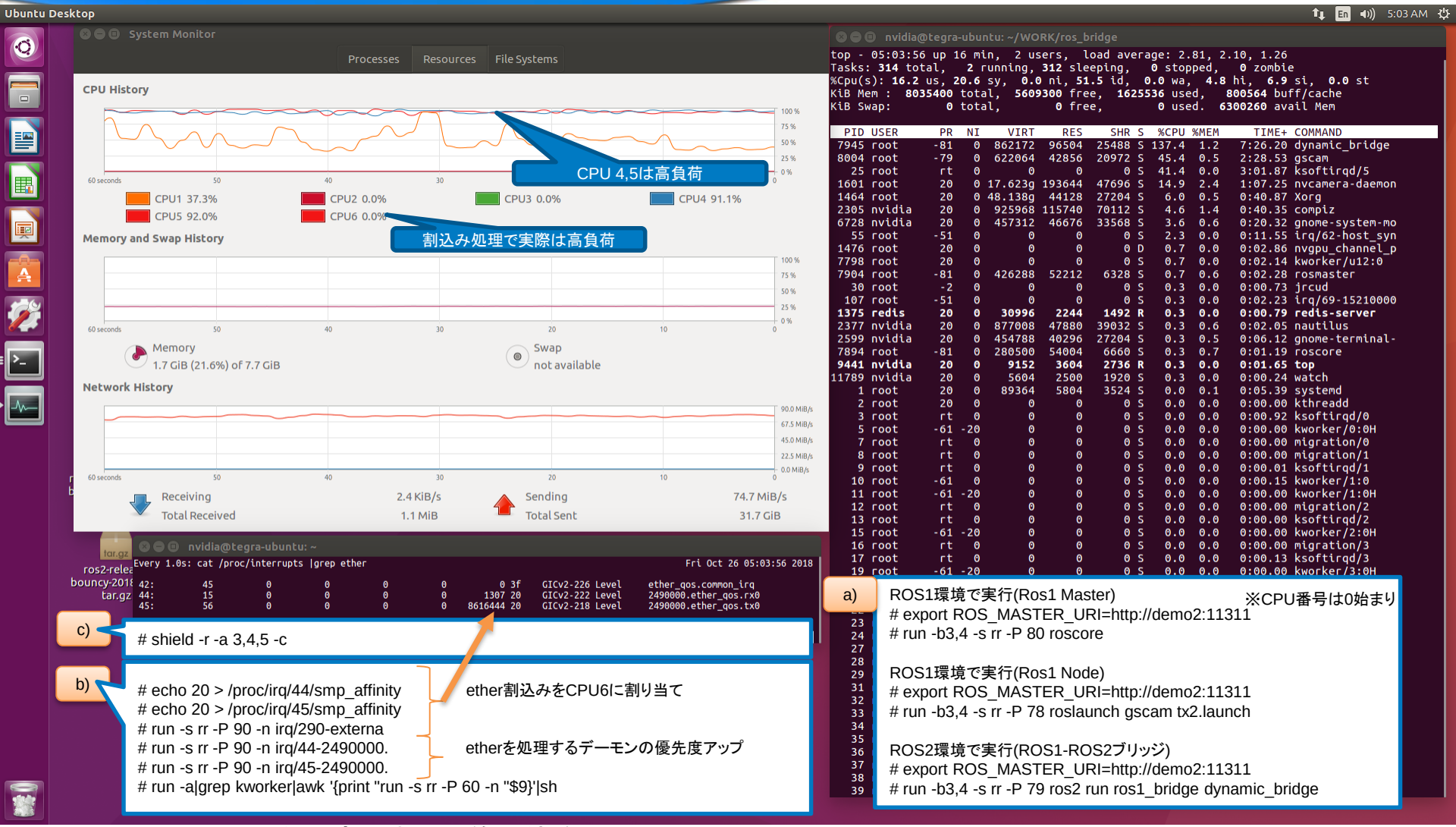
NvidiaカーネルのNetworkトラフィック



RedHawkカーネルのNetworkトラフィック



Networkトラフィックのチューニング



ROS 2.0通信を最適化する最善の戦略は、

a)リアルタイム設定でアプリケーションを構成する、

b)アプリケーション設定に応じてカーネルスレッドを構成する、

c)通信に関与する各マシンのネットワークとCPUの使用を制限する、

TegraSoc & RedHawk導入のメリット

Nvidia　L4Tの長所を損なわずにリアルタイム応用

Jetson

- NVIDIA JetPack SDK
  - TensorRT 3.0 GA
  - cuDNN 7.0.5
  - VisionWorks 1.6
  - CUDA 9.0
  - マルチメディアAPI
    - カメラアプリケーションAPI:libargus
    - センサードライバAPI：V4L2 API
- 深層学習　OSS
  - https://github.com/dusty-nv/
- ROS1/ROS2　OSS
  - https://github.com/peter-moran/jetson_csi_cam
  - https://github.com/ros-drivers/gscam
  - GAZEBO

RedHawk

- JetPackバイナリ互換
  - 左記のソフトウェアをすべて利用可能
- ハードリアルタイムとリソース管理
  - 制御プロセス
  - IOドライバ
  - ネットワーク
  - FBS
- リアルタイム開発ツール
  - NightStarTools
  - SimWB
    - MATLAB/Simlinkと連携
- サービス
  - リアルタイム処理に特化
  - デバイスドライバ開発

ご質問は？

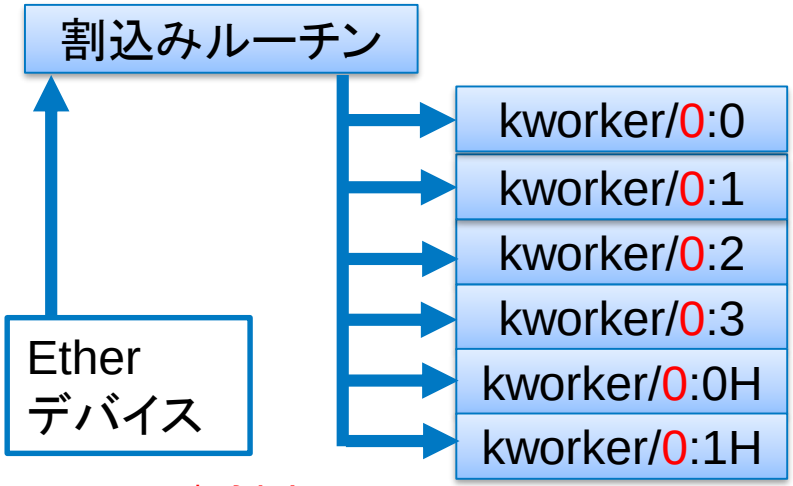
資料編

Jetson TX2/TX2i

割込みの副作用の例

- Etherケーブルを挿抜するとジッターが発生する。
- CPUnで割込み処理を行うkworkerの優先度を上げる
- Kworkerの名称
  - kworker/%u:%d%s (cpu, id, priority)
    kworker/u13:0 のようなuスレッドは現在CPUバインドされていないことを意味する。
    負のnice値(-20)を持つkworkerは、名前に「H」が付いている。
- 例　優先度0:スケジューリング Other

```
$ sudo run -s fifo -P 90 -n kworker/0:0
$ sudo run -s fifo -P 90 -n kworker/0:1
$ sudo run -s fifo -P 90 -n kworker/0:2
$ sudo run -s fifo -P 90 -n kworker/0:3
$ sudo run -s fifo -P 90 -n kworker/0:0H
$ sudo run -s fifo -P 90 -n kworker/0:1H
```



まとめて

```
$ sudo run -a|grep kworker|awk '{print "run -s fifo -P 90 -n "$9}'|sh
```

RedHawk カーネルスレッド

名称(赤字部分なし)	
tegradc.0/*	tegra-iommu
irq/*-gpcdma.*	tegra186-gpc-dma
irq/60-xotg	XUSB OTG Controller
irq/60-3530000.xhci	Xhci USB Controller
irq/22-3460000.sdhi irq/23-3440000.sdhi irq/24-3400000.sdhi irq/256-3400000.sdhi	Tegra Secure Digital Host Controller
irq/64-150c0000.nvcsi irq/65-15700000.vi	Sensor Controller
dhd_dpc dhd_rxf	Broadcom Wireless Driver
irq/55-mc_status	Memory (error) Controller
irq/424-30c0000.watchdog	Watchdog Controller
irq/431-max7762.top	Max77620 Controller
irq/62-host_syncpt irq/63-host_staus	Tegra Graphics Host Syncpoint Integration
kworker/%u:%d%s (cpu, id, priority)	割り込み、タイマー、I / Oなどがある場合に、カーネルの実際の処理の大部分を実行するカーネルワーカースレッド。

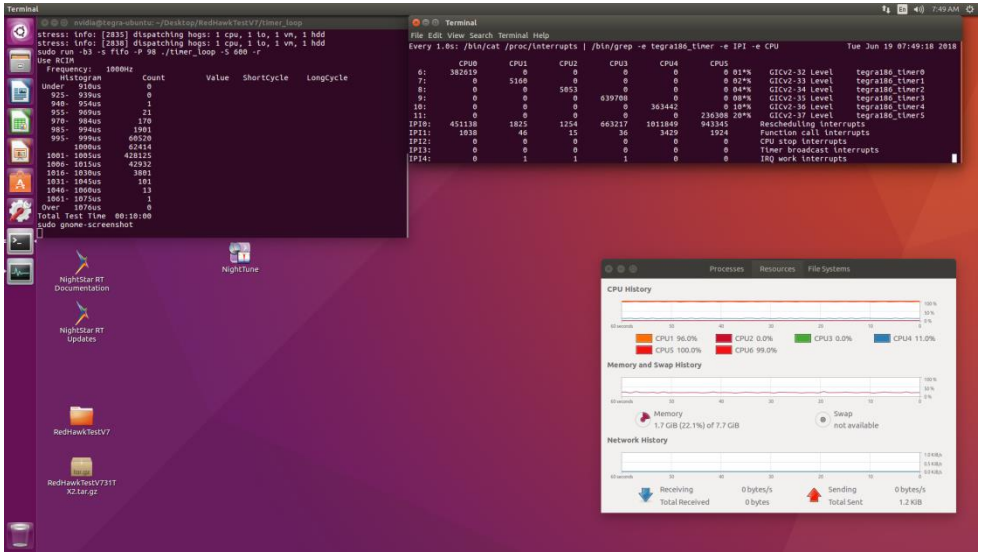
RedHawk カーネルスレッドの優先度(50と99の間)

Pid	Tid	Bias	Actual	CPU	Policy	Pri	Nice	Name
7	7	0x01	0x01	0	fifo	100	0	migration/0
8	8	0x00	0x02	1	fifo	100	0	migration/1
12	12	0x00	0x04	2	fifo	100	0	migration/2
16	16	0x08	0x08	3	fifo	100	0	migration/3
20	20	0x10	0x10	4	fifo	100	0	migration/4
24	24	0x20	0x20	5	fifo	100	0	migration/5
32	32	0x39	0x31	0	fifo	100	0	ltmrd
34	34	0x39	0x31	0	fifo	50	0	irq/55-mc_statu
43	43	0x39	0x31	4	fifo	50	0	irq/424-30c0000
44	44	0x39	0x31	0	fifo	50	0	irq/431-max7762
56	56	0x39	0x31	4	fifo	50	0	irq/62-host_syn
57	57	0x39	0x31	0	fifo	50	0	irq/63-host_sta
101	101	0x39	0x31	3	fifo	1	0	tegradc.0/a
102	102	0x39	0x31	0	fifo	1	0	tegradc.0/b
103	103	0x39	0x31	0	fifo	1	0	tegradc.0/c
104	104	0x39	0x31	0	fifo	1	0	tegradc.0/d
105	105	0x39	0x31	3	fifo	1	0	tegradc.0/e
106	106	0x39	0x31	3	fifo	1	0	tegradc.0/f
107	107	0x39	0x31	3	fifo	1	0	tegradc.0/sl
108	108	0x39	0x31	4	fifo	50	0	irq/69-15210000
112	112	0x39	0x31	5	fifo	50	0	irq/73-gk20a_st
115	115	0x39	0x31	0	fifo	50	0	irq/252-1521000
143	143	0x39	0x31	0	fifo	50	0	irq/54-tegra_rt
150	150	0x39	0x31	0	fifo	50	0	irq/22-3460000.
151	151	0x39	0x31	0	fifo	50	0	irq/77-b150000.
153	153	0x39	0x31	3	fifo	50	0	irq/78-b150000.
155	155	0x39	0x31	4	fifo	50	0	irq/78-b150000.
156	156	0x39	0x31	0	fifo	50	0	irq/52-b000000.
157	157	0x39	0x31	0	fifo	50	0	irq/81-c150000.
158	158	0x39	0x31	4	fifo	50	0	irq/53-d230000.
171	171	0x39	0x31	0	fifo	50	0	irq/92-gpcdma.0
172	172	0x39	0x31	0	fifo	50	0	irq/93-gpcdma.1
173	173	0x39	0x31	0	fifo	50	0	irq/94-gpcdma.2
174	174	0x39	0x31	0	fifo	50	0	irq/95-gpcdma.3
175	175	0x39	0x31	0	fifo	50	0	irq/96-gpcdma.4
176	176	0x39	0x31	0	fifo	50	0	irq/97-gpcdma.5
177	177	0x39	0x31	0	fifo	50	0	irq/98-gpcdma.6
178	178	0x39	0x31	0	fifo	50	0	irq/99-gpcdma.7
179	179	0x39	0x31	0	fifo	50	0	irq/100-gpcdma.
180	180	0x39	0x31	0	fifo	50	0	irq/101-gpcdma.
181	181	0x39	0x31	0	fifo	50	0	irq/102-gpcdma.
182	182	0x39	0x31	0	fifo	50	0	irq/103-gpcdma.
183	183	0x39	0x31	0	fifo	50	0	irq/104-gpcdma.
184	184	0x39	0x31	0	fifo	50	0	irq/105-gpcdma.
185	185	0x39	0x31	0	fifo	50	0	irq/106-gpcdma.
186	186	0x39	0x31	0	fifo	50	0	irq/107-gpcdma.
187	187	0x39	0x31	0	fifo	50	0	irq/108-gpcdma.
188	188	0x39	0x31	0	fifo	50	0	irq/109-gpcdma.
189	189	0x39	0x31	0	fifo	50	0	irq/110-gpcdma.
190	190	0x39	0x31	0	fifo	50	0	irq/111-gpcdma.
191	191	0x39	0x31	0	fifo	50	0	irq/112-gpcdma.
192	192	0x39	0x31	0	fifo	50	0	irq/113-gpcdma.
193	193	0x39	0x31	0	fifo	50	0	irq/114-gpcdma.
194	194	0x39	0x31	0	fifo	50	0	irq/115-gpcdma.
195	195	0x39	0x31	0	fifo	50	0	irq/116-gpcdma.
196	196	0x39	0x31	0	fifo	50	0	irq/117-gpcdma.
197	197	0x39	0x31	0	fifo	50	0	irq/118-gpcdma.
198	198	0x39	0x31	0	fifo	50	0	irq/119-gpcdma.
199	199	0x39	0x31	0	fifo	50	0	irq/120-gpcdma.
200	200	0x39	0x31	0	fifo	50	0	irq/121-gpcdma.
201	201	0x39	0x31	0	fifo	50	0	irq/122-gpcdma.
202	202	0x39	0x31	0	fifo	50	0	irq/123-gpcdma.
203	203	0x39	0x31	0	fifo	50	0	irq/23-3440000.
207	207	0x39	0x31	0	fifo	50	0	irq/24-3400000.
209	209	0x39	0x31	4	fifo	1	0	mmcqd/0
211	211	0x39	0x31	3	fifo	1	0	mmcqd/0boot0
213	213	0x39	0x31	4	fifo	1	0	mmcqd/0boot1
215	215	0x39	0x31	0	fifo	1	0	mmcqd/0rpmb
216	216	0x39	0x31	0	fifo	50	0	irq/256-3400000
219	219	0x39	0x31	0	fifo	50	0	irq/64-150c0000
221	221	0x39	0x31	0	fifo	50	0	irq/60-3530000.
222	222	0x39	0x31	0	fifo	50	0	irq/34-3210000.
224	224	0x39	0x31	0	fifo	50	0	irq/35-c260000.
226	226	0x39	0x31	0	fifo	50	0	irq/36-3240000.
229	229	0x39	0x31	0	fifo	50	0	irq/60-xotg
234	234	0x39	0x31	0	fifo	50	0	irq/65-15700000
640	640	0x01	0x01	0	fifo	50	0	dhd_dpc
641	641	0x39	0x31	0	fifo	50	0	dhd_rxf
3	3	0x01	0x01	0	rr	99	0	ksoftirqd/0
9	9	0x00	0x02	1	rr	99	0	ksoftirqd/1
13	13	0x00	0x04	2	rr	99	0	ksoftirqd/2
17	17	0x08	0x08	3	rr	99	0	ksoftirqd/3
21	21	0x10	0x10	4	rr	99	0	ksoftirqd/4
25	25	0x20	0x20	5	rr	99	0	ksoftirqd/5
30	30	0x39	0x31	0	rr	1	0	jrcud
2063	2094	0x39	0x31	5	rr	5	0	pulseaudio
2063	2107	0x39	0x31	0	rr	5	0	pulseaudio
2063	2108	0x39	0x31	3	rr	5	0	pulseaudio
2068	2079	0x39	0x31	0	rr	99	1	rtkit-daemon

GPU副作用の軽減

- Xorgとnvgpu_channel_pollの優先度を上げると、安定する
- nvgpu_channel_poll:GPUとの通信路(FIFO)を管理

```
$ sudo run -b0 -s rr -P 51 -n nvgpu_channel_p
$ sudo run -b0 -s rr -P 51 -n Xorg -L a
$ run -a|grep -e Xorg -e gpu
1048    1048    0x01    0x01      0      rr         51     0       Xorg
1064    1064    0x01    0x01      0      rr         51     0       nvgpu_channel_p
```

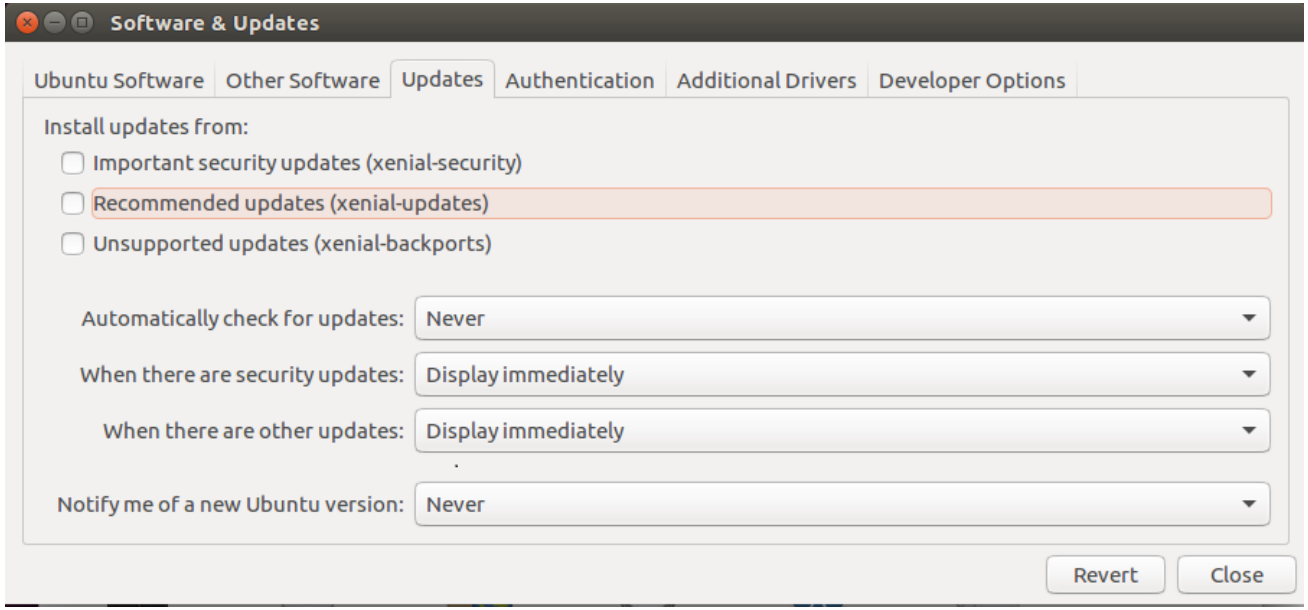


Tegra Fatal Error(Over Load)

```
/usr/bin/lspci
00:01.0 PCI bridge: NVIDIA Corporation Device 10e5 (rev a1) (prog-if 00 [Normal decode])
            Control: I/O+ Mem+ BusMaster+ SpecCycle- MemWINV- VGASnoop- ParErr- Stepping- SERR- FastB2B-
DisINTx-
            Status: Cap+ 66MHz- UDF- FastB2B- ParErr- DEVSEL=fast >TAbort- <TAbort- <MAbort- >SERR- <PERR-
INTx-
            Latency: 0
            Interrupt: pin A routed to IRQ 388
            Bus: primary=00, secondary=01, subordinate=02, sec-latency=0
            I/O behind bridge: 00001000-00001fff
            Memory behind bridge: 50100000-501fffff
            Prefetchable memory behind bridge: 0000000058000000-00000000580fffff
            Secondary status: 66MHz- FastB2B- ParErr- DEVSEL=fast :

:

            Kernel driver in use: pcieport

$ /sbin/lsirq –s
388         00:01.0 NVIDIA Corporation Unknown device (rev ff)

$ dmesg|grep pcieport
:

kernel: pcieport 0000:00:01.0: AER: Device recovery failed
kernel: pcieport 0000:00:01.0: AER: Uncorrected (Non-Fatal) error received: id=0020
kernel: pcieport 0000:00:01.0: PCIe Bus Error: severity=Uncorrected (Non-Fatal), type=Transaction Layer,
id=0008(Requester ID)
kernel: pcieport 0000:00:01.0:   device [10de:10e5] error status/mask=00004000/00000000
kernel: pcieport 0000:00:01.0:    [14] Completion Timeout     (First)
kernel: pcieport 0000:00:01.0: broadcast error_detected message
```

AutoUpdate/Upgradeの設定を解除する

- Ubuntu 16.04を起動した時と毎日6時と18時（ランダムなディレイあり）にAutoUpdate/Upgradeが実行がされます。
- デフォルトでは、セキュリティアップグレードのみ実行され、自動再起動は実行されません。
- ネットワークが未接続の状態で実行されると負荷100%になってジッターの原因になります。
- GUIのSoftware&Updateの画面から設定できます。



Ja_JPのサポート

- サポートされている言語の確認

  $ less /usr/share/i18n/SUPPORTED
- インストールされている言語の確認

  $ locale -a
- ja_JP.UTF-8を追加

  $ sudo locale-gen ja_JP.UTF-8
- 日本語環境をインストール

  $ sudo apt-get install language-pack-ja
- リストを再生成する

  $ sudo dpkg-reconfigure locales
- 設定はGUIから

⇒漢字入力は出来ない

Jetson TX2クローニング(1)

- L4Tツールを使用すると、eMMCパーティションをバックアップ/リストアすることで、Jetson TX2のクローニングをすることができます。
- JetsonはUSB経由でリモートPCに接続し、リカバリモードに入っている必要があります。
- イメージのバックアップ
  - ホストPC上のL4Tインストールパッケージが入っているディレクトリにcdしてください。以下のコマンドは、TX2のeMMCイメージを-Gで指定されたファイルに保存します。

```
$ cd ~/64_TX2/Linux_for_Tegra
$ sudo ./flash.sh -r -k APP -G backup.img jetson-tx2 mmcblk0p1
```

- この場合、backup.imgとbackup.img.rawというファイルを生成します

出典　https://devtalk.nvidia.com/default/topic/1000105/?comment=5111893

Jetson TX2クローニング(2)

- バックアップイメージをフラッシュするディレクトリにコピーする
  - ソースデバイスから完全なイメージを含む.rawファイルをコピーします。

```
$ cd ~/64_TX2/Linux_for_Tegra
$ sudo mv bootloader/system.img system.img.original
$ sudo cp backup.img.raw bootloader/system.img
```

- イメージのリストア
  - 別のシリアル番号を持つ複数のユニットを復元するには、上記のイメージを "system.img"として保存し、ヘッドL4Tフラッシュスクリプトflash.shを-rオプションとともに使用します( )バニライメージを最初から再構築することなくバックアップされたsystem.imgを再利用するため)

```
$ sudo ./flash.sh -r -k APP jetson-tx2 mmcblk0p1
```

sdから起動する(1)

- ルートファイルシステムをSDカードにコピーし、SDカードから起動するチュートリアルです。
- ルートファイルシステムがSDカード上にある場合は、以下のスクリプトを実行して、Jetson TX1およびTX2開発キットに記載されているデバイスをフラッシュしてください。
  - sudo ./flash.sh <プラットフォーム名> mmcblk1p1
- rootfsを配置するデバイスを選ぶ
  - これらの手順を使用して、ファイルシステムをTegraデバイスにコピーします。
  1. 内部EMMCを使用している場合は、ブートローダとカーネルのフラッシュをスキップしてください。
  2. ルートファイルシステムに外部ストレージデバイスを使用する場合は、次の手順を使用して、ファイルシステムを外部ストレージデバイスにコピーします。
  - rootfsデバイスをホストシステムに接続します。デバイスがExt4としてフォーマットされていない場合は、次のコマンドを入力してExt4ファイルシステムでフォーマットします。

$ sudo mkfs.ext4 / dev / sd <port> <device_number>
場所：
<port>は、デバイスがマウントされているポートです。
<device_number>は、ポートに接続されているデバイスのデバイス番号です。
dmesgコマンドを使用してポートを判別できます。

sdから起動する(2)

- 必要に応じて、次のコマンドを使用してデバイスをマウントします。
  sudo mount / dev / sdX1 <mntpoint>
  <mntpoint>はrootfsデバイスのホストシステム上のマウントポイントです。
- ファイルシステムをコピーします。LDK_ROOTFS_DIRが設定されている場合は、次のコマンドを実行します。
  $ cd $ {LDK_ROOTFS_DIR}
  $ sudo cp -a * <mntpoint> && sync
- 設定されていない場合は、次のコマンドを実行して、リリースに含まれているrootfsディレクトリをコピーします。
  $ cd <your_L4T_root> / Linux_for_Tegra / rootfs
  $ sudo cp -a * <mntpoint> && sync
- コンテンツを外部ディスクまたはデバイスにコピーした後、ディスクをアンマウントし、ターゲットのTegraデバイスに接続します。
- TX2にSDカードとプラグインSDカードのプラグを抜く
- sdcardのexlinux.confを変更して、デバイスを起動します。
  / dev / mmcblk0p1を/ dev / mmcblk1p1（/boot/extlinux/extlinux.conf）に変更してください。
- デバイスは/ dev / mmcblk1p1を使用して起動しています

ログイン時にユーザー特権を与える

- /etc/security/capability.confファイルをrootユーザとして編集し、次の行をファイルの最後に追加します

```
role demouser cap_sys_nice
user nvidia demouser
```

- /etc/pam.d/sshdファイルをrootユーザとして編集し、最後のセッションエントリの後、@include行の前に次の行を追加します。

```
session required /lib/aarch64-linux-gnu/security/pam_capability.so
```

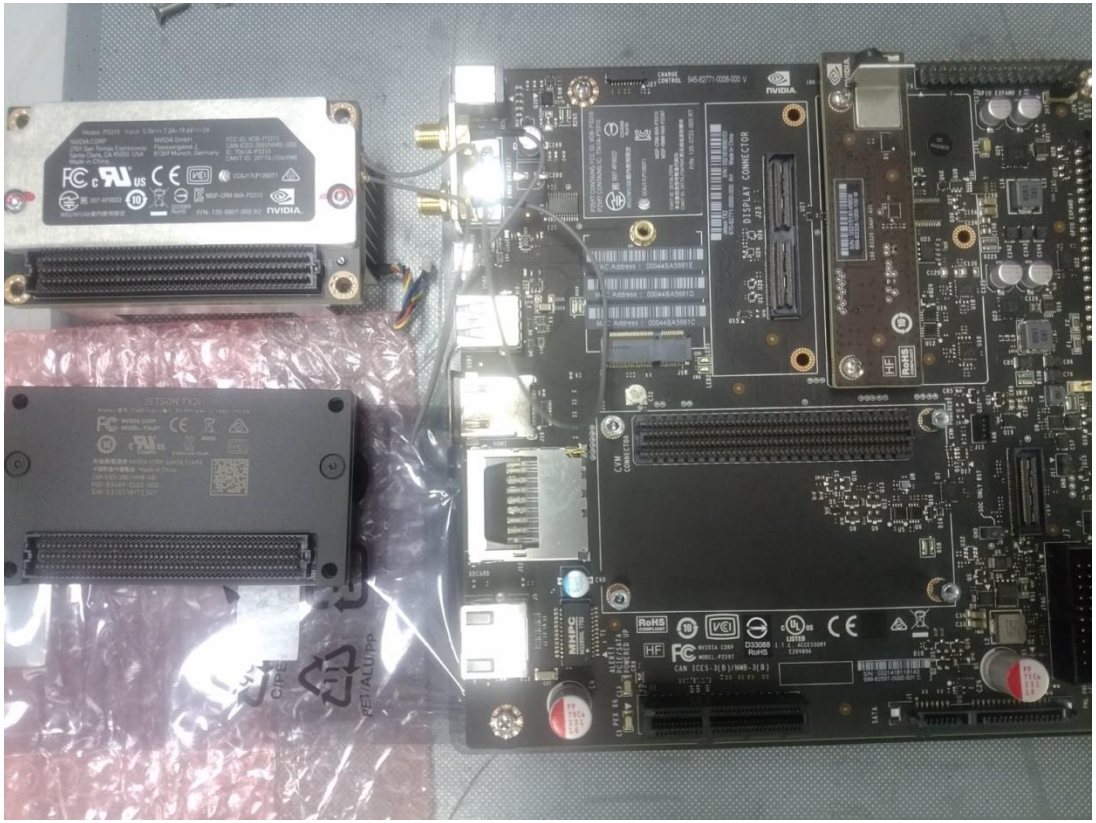
- 同様に/etc/pam.d/lightdmを変更して、グラフィカルログイン時にユーザに自動的に機能を許可することができます。
- 機能プラグインが正しく設定されていることを確認する

```
getpcaps $$
```

- 次のような出力が表示されます。

```
Capabilities for `14554’: = cap_sys_nice+eip
```

Jetson Tx2i



TX2iとTX2の比較

Feature	Jetson TX2i	Jetson TX2
LPDDR4	8 GB, 1600 MHz インダストリアルグレードECC付き	8 GB, 1866 MHzコマーシャルグレード
DRAM ECC	サポート(SEC-DED)	非サポート
eMMC	32GB インダストリアルグレード	32 GB コマーシャルグレード
PMIC	MAX20024 Industrial Grade	MAX77620
Input Voltage	9V to 19.0V	5.5V to 19.6V
SDMMC	デュアルインターフェイス(SD Card and SDIO)	シングルインターフェース(SD Card or SDIO)
Wireless	オンモジュールソリューションなし	BCM4354 on module
VIN Monitor	スーパーバイザはVDD_INをモニタし、VDD_INが有効でない場合はVIN_PWR_BAD#をローに駆動します	Tegra X2のSOC_THERMへの出力でVDD_INをモニタするコンパレータ
(B49)	MOD_PWR_CFG_ID	RSVD
(B48)	SYS_WAKE#：Tegra X2 GPIO_SYS_4ピンに接続し、電源ボタン割り込みとSC7ウェイクアップを行います。	RSVD
Auto-Power-On	POWER_BTN#ピンは未接続 モジュールのプルアップにより信号がハイになり、入力時にPMICレベルに敏感なため電源がオンになります	CHARGER_PRSNT#をGNDに接続することでサポート

TX2iには不要なデーモン

- wpa_supplicant はクロスプラットフォームのサプリカントで WEP, WPA, WPA2 をサポートしています。

ディフォルト

```
$ sudo systemctl stop wpa_supplicant.service
$ sudo systemctl disable wpa_supplicant.service
```

Active

```
$ sudo systemctrl stop bluetooth.service
$ sudo systemctl disable bluetooth.service
```

Inactive

製品名	Jetson™ TX2i	Jetson™ TX2
1 Gigabit Ethernet	✔	✔
802.11ac WLAN	-	✔
Bluetooth	-	✔

TX2iの電源

- Jetson TX2iがP2597 B02 / B04キャリアボードに搭載されている場合、Jetson TX2iモジュールのPMICとは異なるため、主電源が接続されるとすぐに電源がオンになります。
- 電源ボタンを押すと、モジュールとシステムの電源がオフになります。電源ボタンを使用してシステムをスリープモードにすることはできません。
- 具体的には、電源を切ることはできず、自動的に電源が入ります。

TX2iの電源

```
$ sudo shutdown -h 0
[sudo] password for nvidia:
Shutdown scheduled for Thu 2018-07-26 01:09:57 UTC, use 'shutdown -c' to cancel.
nvidia@tegra-ubuntu:~$ tegradc 15210000.nvdisplay: blank - powerdown
PD DISP2 index4 DOWN
PD DISP1 index3 DOWN
PD DISP0 index2 DOWN
tegradc 15210000.nvdisplay: unblank
:
:
>--eqos_shutdown
therm-fan-est: shutting down
gk20a 17000000.gp10b: shutting down
gk20a 17000000.gp10b: shut down complete
PD DISP2 index4 DOWN
PD DISP1 index3 DOWN
PD DISP0 index2 DOWN
i2c i2c-4: Bus is shutdown down...
late_shutdown started
tegra-i2c 31e0000.i2c: Bus is shutdown down..
tegra-i2c c250000.i2c: Bus is shutdown down..
tegra-i2c 31c0000.i2c: Bus is shutdown down..
tegra-i2c 31b0000.i2c: Bus is shutdown down..
tegra-i2c 3190000.i2c: Bus is shutdown down..
tegra-i2c 3180000.i2c: Bus is shutdown down..
tegra-i2c c240000.i2c: Bus is shutdown down..
tegra-i2c 3160000.i2c: Bus is shutdown down..
Disabling non-boot CPUs ...
CPU1: shutdown
psci: CPU1 killed.
CPU2: shutdown
psci: CPU2 killed.
CPU3: shutdown
psci: CPU3 killed.
CPU4: shutdown
psci: CPU4 killed.
CPU5: shutdown
psci: CPU5 killed.
reboot: Power down
max77620-power max20024-power: Powering off system
max77620-power max20024-power: Avoid PMIC power off

[0009.505] I> Welcome to MB2(TBoot-BPMP)(version: 01.00.160913-t186-M-00.00-mobile-c4328dc3)
```

自動的に再起動する

Tx2iのECCドライバ

- シングルビットエラー割り込み、ダブルビットエラー 割り込が発生。

```
$ dmesg|grep -i ecc
[    0.226929] ramoops: attached 0x200000@0x237080000, ecc: 0/0
[    0.341560] ecc-err: dram ecc enabled-MC_ECC_CONTROL:0x0000000d
[    0.341641] ecc-err: ecc_err_irq = 56
[    0.341754] ecc-err: EMC_INTMASK = 0x00001c00
[    0.341774] ecc-err: EMC_NONCRITICAL_INTMASK = 0x00000000
[    0.341795] ecc-err: EMC_CRITICAL_INTMASK = 0x00000000
[    3.836894] **** A57 ECC: Enabled

$ cat /proc/interrupts |grep ecc
56: 0 0 0 0 0 0 3f    GICv2-256 Level     ecc_irq_status
$ sudo grep "" /proc/irq/56/*
/proc/irq/56/affinity_hint:00
/proc/irq/56/ecc_irq_status: Is a directory
/proc/irq/56/node:0
/proc/irq/56/smp_affinity:3f user 39 actual
/proc/irq/56/smp_affinity_list:0-5 user 0,3-5 actual
/proc/irq/56/spurious:count 0
/proc/irq/56/spurious:unhandled 0
/proc/irq/56/spurious:last_unhandled 0 ms
```

ECCエラーメッセージ

- $ fgrep ecc_err_pr ./kernel/t18x/drivers/platform/tegra/mc/mcerr_ecc_t18x.c

- ecc_err_pr("EMC_ECC_ERR_REQ = 0x%08x¥n", log->emc_ecc_err_req);
- ecc_err_pr("EMC_ECC_ERR_SP0 = 0x%08x¥n", log->emc_ecc_err_sp0);
- ecc_err_pr("EMC_ECC_ERR_SP1 = 0x%08x¥n", log->emc_ecc_err_sp1);
- ecc_err_pr("EMC_ECC_ERR_ADDR = 0x%08x¥n", log->ecc_err_addr);
- ecc_err_pr("D.X.R.B.C.S.L:%d.%d.0x%x.%d.0x%x.%d.0¥n", log->ecc_err_dev,
- ecc_err_pr("demand scrub timeout, addr = 0x%x¥n",
- ecc_err_pr("Demand scrubbing for correctable DRAM ECC Error¥n");
- ecc_err_pr("SBE reported for read from HW scrubber¥n");
- ecc_err_pr("POISON Bit ERR, addr:0x%016llx¥n", addr);
- ecc_err_pr("Data bit DBE ERR , addr:0x%016llx¥n", addr);
- ecc_err_pr("ecc bit DBE ERR , addr:0x%016llx¥n", addr);
- ecc_err_pr("Data bit SBE ERR, addr:0x%016llx¥n", addr);
- ecc_err_pr("ECC bit SBE ERR, addr:0x%016llx¥n", addr);
- ecc_err_pr("BUFF OVER FLOW ERR¥n");

customカーネルの作成
/boot/extlinux/extlinux.conf
シリアルコンソールセットアップ
シリアルコンソールの起動操作
ソースコードツリーの一部を動的モジュールとしてコンパイルする
動的カーネルモジュールの情報

カスタムカーネルとドライバ

customカーネルの作成(dtbファイルが必要な場合)

```
$ sudo ./ccur-config -k custom -n trace
$ sudo make REDHAWKFLAVOR=-custom zImage
$ sudo make REDHAWKFLAVOR=-custom dtbs
$ sudo make REDHAWKFLAVOR=-custom modules
$ sudo make REDHAWKFLAVOR=-custom modules_install
$ sudo make REDHAWKFLAVOR=-custom install
```

この時点で、/boot下には下記のファイルが生成されているが、

```
/boot/config-4.4.38-rt49-r28.2.1-RedHawk-7.3.2-custom
/boot/initrd.img-4.4.38-rt49-r28.2.1-RedHawk-7.3.2-custom
/boot/Image-4.4.38-rt49-r28.2.1-RedHawk-7.3.2-custom
/boot/System.map-4.4.38-rt49-r28.2.1-RedHawk-7.3.2-custom
```

DTBディレクトリは生成されていないので、arch/arm64/boot/dts/*dtbファイルをマニュアルでインストールする

```
$ sudo mkdir /boot/dtb-4.4.38-rt49-r28.2.1-RedHawk-7.3.2-custom
$ sudo cp arch/arm64/boot/dts/*dtb /boot/dtb-4.4.38-rt49-r28.2.1-RedHawk-7.3.2-custom
```

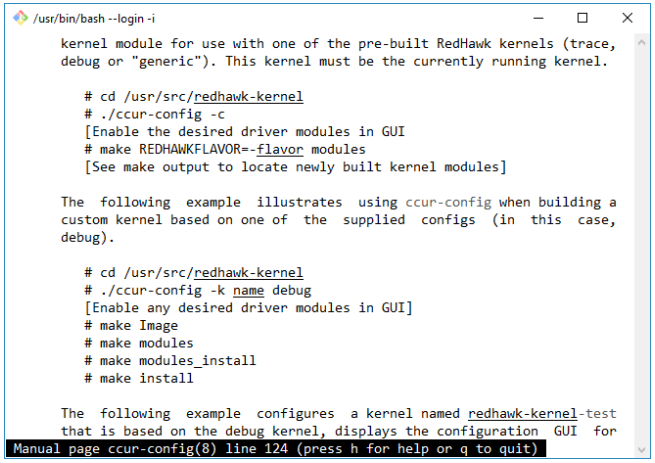
customカーネルの作成(dtbファイルが不必要な場合)

```
$ sudo ./ccur-config -k custom -n debug
$ sudo make Image
$ sudo make modules
$ sudo make modules_install
$ sudo make install
```

この時点で、/boot下には下記のファイルが生成されている、

```
/boot/config-4.4.38-rt49-r28.2.1-RedHawk-7.3.2-custom
/boot/initrd.img-4.4.38-rt49-r28.2.1-RedHawk-7.3.2-custom
/boot/Image-4.4.38-rt49-r28.2.1-RedHawk-7.3.2-custom
/boot/System.map-4.4.38-rt49-r28.2.1-RedHawk-7.3.2-custom
/boot/dtb-4.4.38-rt49-r28.2.1-RedHawk-7.3.2-debug を共有する
```

```
$ man ccur-config
```



/boot/extlinux/extlinux.conf

```
PROMPT 1
TIMEOUT 30
DEFAULT redhawk-trace
MENU TITLE p2771-0000 eMMC boot options
LABEL primary
      MENU LABEL primary kernel
      LINUX /boot/Image
      APPEND ${cbootargs} root=/dev/mmcblk0p1 rw rootwait rootfstype=ext4
LABEL redhawk-trace
      MENU LABEL trace kernel
      LINUX /boot/Image-4.4.38-rt49-r28.2.1-RedHawk-7.3.2-trace
      INITRD /boot/initrd.img-4.4.38-rt49-r28.2.1-RedHawk-7.3.2-trace
      APPEND ${cbootargs} root=/dev/mmcblk0p1 rw rootwait rootfstype=ext4
LABEL redhawk-custom
      MENU LABEL custom kernel
      LINUX /boot/Image-4.4.38-rt49-r28.2.1-RedHawk-7.3.2-custom
      INITRD /boot/initrd.img-4.4.38-rt49-r28.2.1-RedHawk-7.3.2-custom
      APPEND ${cbootargs} root=/dev/mmcblk0p1 rw rootwait rootfstype=ext4
```

安全に起動するようになってから変更する

RedHawk Linux デバイスドライバ開発

- extmem

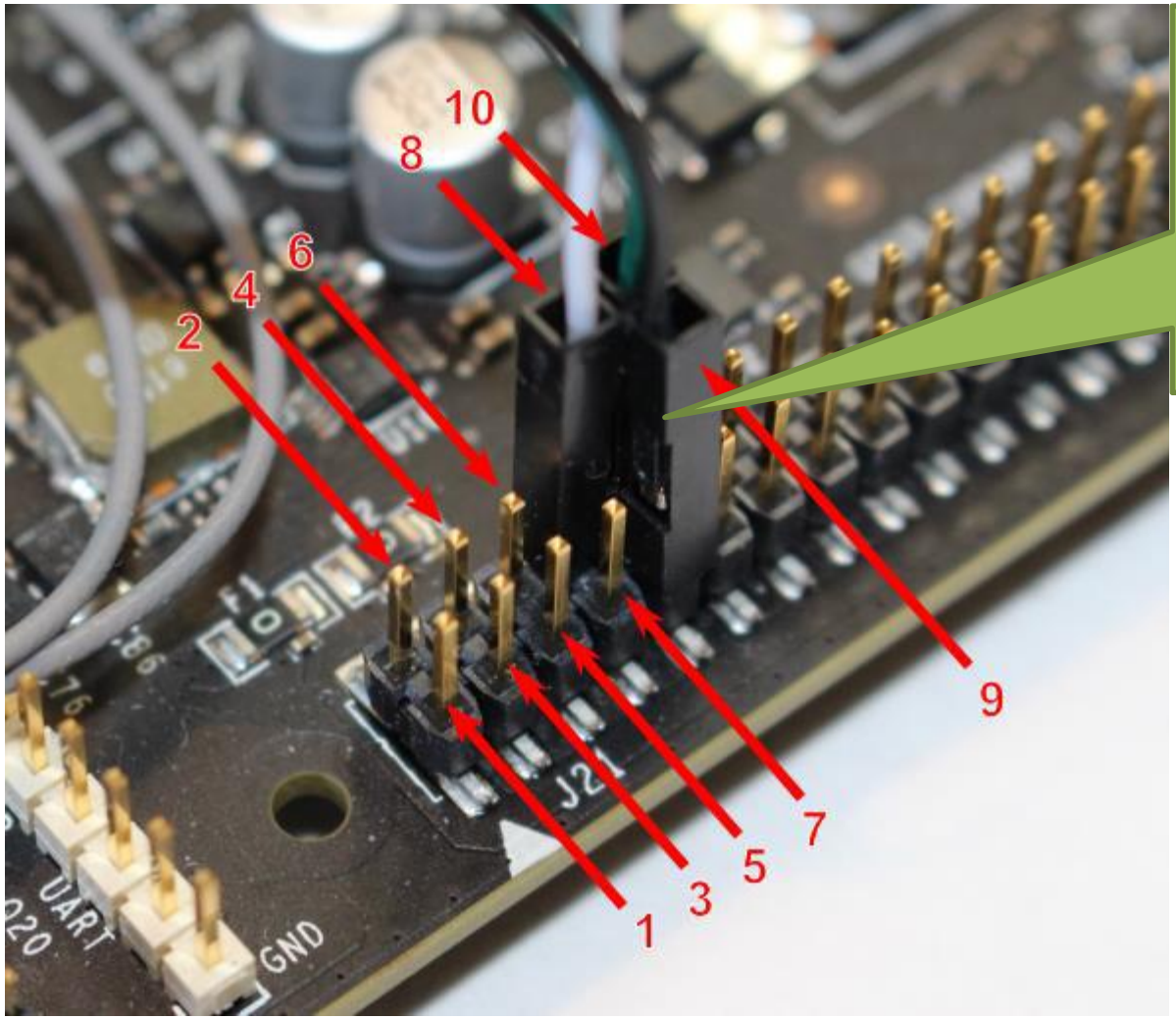
AIO-163202F-PE (Note : No Support DMA)	Multi Function AD 16bit SE:32ch DI:16ch DA 16bit 2ch DIO 8ch Counter 32bit 2ch	AD:2μsec/ch (Max.) バイポーラ ±10V、±5V、±2.5V またはユニポーラ 0 - +10V、0 - +5V、0 - +2.5V DA:10μsec(Max.) バイポーラ ±10V、±5V、±2.5V、±1.25V またはユニポーラ 0 - +10V、0 - +5V、0 - +2.5V DIO: 非絶縁入力 8 点(LVTTL レベル正論理) 非絶縁出力 8 点 (LVTTL レベル 正論理)
AO-1616L-LPE	DA:16 bit 16ch DIO 4ch counter 32bit 1ch	バイポーラ ±10V 変換速度 10μsec 非絶縁入力 4 点 (LVTTL レベル 正論理) 非絶縁出力 4 点 (LVTTL レベル 正論理)
DI-128RL-PC PI-128L(PCI)H DI-128T2-PCI DI-128L-PE DI-128T-PE	DI: 128ch Device IDを変更することによって同一Access Libraryで対応	PCI 絶縁型逆コモンタイプ デジタル入力ボード PCI 絶縁型デジタル入力ボード PCI 非絶縁型デジタル入力ボード PCIExpress 絶縁型デジタル入力ボード PCIExpress 非絶縁型デジタル入力ボード
DO-128RL-PCI PO-128L(PCI)H DO-128T2-PCI DO-128L-PE DO-128T-PE	DO: 128ch Device IDを変更することによって同一Access Libraryで対応	PCI 絶縁型逆コモンタイプ デジタル出力ボード PCI 絶縁型デジタル出力ボード PCI 非絶縁型デジタル出力ボード PCIExpress 絶縁型デジタル出力ボード PCIExpress 非絶縁型デジタル入力ボード

RedHawk Linux デバイスドライバ開発

- extmem

PIO-32/32L(PCI)H PIO-32/32RL(PCI)H PIO-32/32T(PCI)H PIO-32/32B(PCI)V PIO-32/32H(PCI)H PIO-32/32F(PCI)H DIO-3232L-PE DIO-3232T-PE DIO-3232B-PE DIO-3232F-PE	DIO: 32ch Device IDを変更することによって同一Access Libraryで対応	PCI 絶縁型デジタル入出力ボード PCI 絶縁型逆コモンタイプデジタル入出力ボード 非絶縁型デジタル入出力ボード 絶縁型デジタル入出力ボード(電源内蔵) 絶縁型デジタル入出力ボード 高速絶縁型デジタル入出力ボード 絶縁型デジタル入出力ボード 非絶縁型デジタル入出力ボード 絶縁型デジタル入出力ボード(電源内蔵) 高速絶縁型デジタル入出力ボード
PIO-64/64L(PCI)H DIO-6464T2-PCI DIO-6464L-PE DIO-6464T-PE	DIO: 64ch Device IDを変更することによって同一Access Libraryで対応	PCI 絶縁型デジタル入出力ボード PCI 非絶縁型デジタル入出力ボード PCIExpress 絶縁型デジタル入出力ボード PCIExpress 非絶縁型デジタル入出力ボード
PMC-5565PIORC PCIE-5565PIORC PCIE-5565RC PCI-5565PIORC	リフレクティブメモリ DMA Support	PMC 128 or 256MB PCI Express 128 or 256MB PCI Express 128 or 256 MB PCI 128 or 256MB

シリアルコンソールセットアップ



- 8bits パリティ無し
- 115200 bit/sec
- ハードフロー制御なし
- ソフトフロー制御あり

シリアルコンソールの起動操作

```
$ sudo reboot
----------------------------------------
Select kernel with serial console
----------------------------------------
U-Boot 2016.07-g9c3b9a4 (Mar 01 2018 - 20:41:10 -0800)

TEGRA186
Model: NVIDIA P2771-0000-500
DRAM:  7.8 GiB
MC:   Tegra SD/MMC: 0, Tegra SD/MMC: 1
*** Warning - bad CRC, using default environment

In:    serial
Out:   serial
Err:   serial
Net:   eth0: ethernet@2490000
Hit any key to stop autoboot:  0
MMC: no card present
switch to partitions #0, OK
mmc0(part 0) is current device
Scanning mmc 0:1...
Found /boot/extlinux/extlinux.conf
Retrieving file: /boot/extlinux/extlinux.conf
734 bytes read in 93 ms (6.8 KiB/s)
p2771-0000 eMMC boot options
1:      primary kernel
2:      trace kernel
3:      custom kernel
Enter choice: 3  3
3:      custom kernel
```

3を選ぶ

```
Retrieving file: /boot/initrd.img-4.4.38-rt49-r28.2.1-RedHawk-7.3.2-custom
9494754 bytes read in 493 ms (18.4 MiB/s)
Retrieving file: /boot/Image-4.4.38-rt49-r28.2.1-RedHawk-7.3.2-custom
15525984 bytes read in 649 ms (22.8 MiB/s)
append: root=/dev/mmcblk0p1 rw rootwait console=ttyS0,115200n8 console=tty0 OS=l4t fbcon=map:0 net.ifnames=0
memtype=0 video=tegrafb no_console_suspend=1 earlycon=uart8250,mmio32,0x03100000
nvdumper_reserved=0x2772e0000 gpt tegra_fbmem2=0x800000@0x969ec000 lut_mem2=0x2008@0x969e9000
tegraid=18.1.2.0.0 tegra_keep_boot_clocks maxcpus=6 boot.slot_suffix= boot.ratchetvalues=0.2.1
androidboot.serialno=0324717065224 bl_prof_dataptr=0x10000@0x277040000 sdhci_tegra.en_boot_part_access=1
root=/dev/mmcblk0p1 rw rootwait rootfstype=ext4
## Flattened Device Tree blob at 92000000
   Booting using the fdt blob at 0x92000000
   reserving fdt memory region: addr=80000000 size=10000
   Using Device Tree in place at 0000000092000000, end 0000000092049273
Starting kernel ...

Booting Linux on physical CPU 0x100
:
RedHawk Linux release 7.3.2 (Wonka)
 on an aarch64

tegra-ubuntu login: nvidia (automatic login)

Last login: Tue Jun 12 02:23:12 UTC 2018 on ttyS0
Welcome to Ubuntu 16.04 LTS (GNU/Linux 4.4.38-rt49-r28.2-RedHawk-7.3.2-custom aarch64)

 * Documentation:  https://help.ubuntu.com/

513 packages can be updated.
269 updates are security updates.

fuse init (API version 7.23)
nvidia@tegra-ubuntu:~$ tegradc 15210000.nvdisplay: blank - powerdown
PD DISP2 index4 DOWN
PD DISP1 index3 DOWN
PD DISP0 index2 DOWN
tegradc 15210000.nvdisplay: unblank
PD DISP0 index2 UP
PD DISP1 index3 UP
PD DISP2 index4 UP
Parent Clock set for DC plld2
tegradc 15210000.nvdisplay: hdmi: pclk:148500K, set prod-setting:prod_c_150M
tegradc 15210000.nvdisplay: unblank

nvidia@tegra-ubuntu:~$
```

ソースコードツリーの一部を動的モジュールとしてコンパイルする

$ sudo make -C */lib/modules/*`uname -r`/build ¥

SUBDIRS=ターゲットソースコードツリーパス ¥

REDHAWKFLAVOR=`cat /proc/ccur/flavor` ¥

 -e "CONFIG_XXXX=m" module module_install
$ sudo depmod

XXXは、Kconfigに定義されているもので、カーネルソースコードツリーの.configファイルに定義されmake時に利用されている。

=mは動的、=yは静的モジュール(自動的にGPLになる)

例えば

-e "CONFIG_RCIM=m"

動的カーネルモジュールの情報

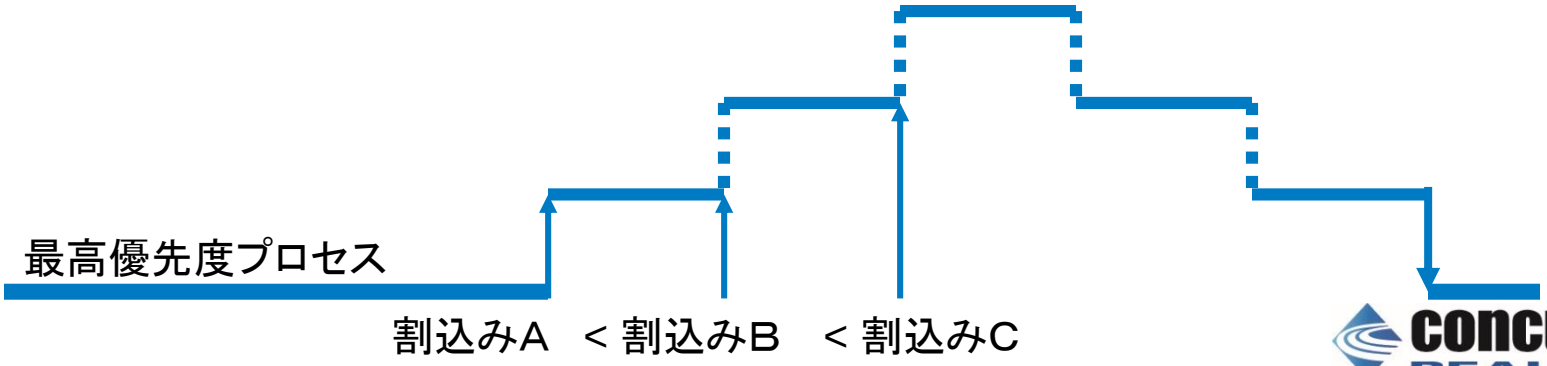
```
$ lsmod
Module                  Size  Used by
fuse                   85350  2
vtop                    2105  0
rcim_emu                6614  0
rcim                   75116  0
bcmdhd                7590901  0
pci_tegra              68117  0
bluedroid_pm            9237  0
ext4                  380890  1
mbcache                10743  1 ext4
jbd2                   71665  1 ext4
dm_mod                102339  0
broadcom                9596  0
bcm_phy_lib             2656  1 broadcom
ahci_tegra             19377  0
libahci_platform        7733  1 ahci_tegra
libahci                28492  2 libahci_platform,ahci_tegra
libata                182452  3 libahci,libahci_platform,ahci_tegra
eqos                  135820  1
ptp                    12437  1 eqos
pps_core                8236  1 ptp

$ modinfo rcim
filename:       /lib/modules/4.4.38-rt49-r28.2.1-RedHawk-7.3.2-trace/kernel/drivers/char/rcim/rcim.ko
description:    Multicard RCIM Driver
author:         Concurrent Real-Time
license:        GPL
alias:          pci:v000010B5d00008845sv*sd*bc*sc*i*
alias:          pci:v00001542d00009260sv*sd*bc*sc*i*
alias:          pci:v00001542d00009271sv*sd*bc*sc*i*
depends:
intree:         Y
vermagic:       4.4.38-rt49-r28.2.1-RedHawk-7.3.2-trace SMP preempt mod_unload aarch64
parm:           nomsi:int
$ modinfo rcim_emu
filename:       /lib/modules/4.4.38-rt49-r28.2.1-RedHawk-7.3.2-trace/kernel/drivers/char/rcim-emu/rcim-emu.ko
description:    RCIM RTC emulator
author:         Concurrent Real-Time
license:        GPL
depends:
intree:         Y
vermagic:       4.4.38-rt49-r28.2.1-RedHawk-7.3.2-trace SMP preempt mod_unload aarch64
```

多重割込みとプリエンプション
PREEMPTカーネルと標準カーネル
PREEMPT vs PREEMPT_RT
ダイレクトIOプログラム
プリアロケート・グラフィックス・バッファ
POSIX-(REALTIME)APIとTHREAD

リアルタイム技術

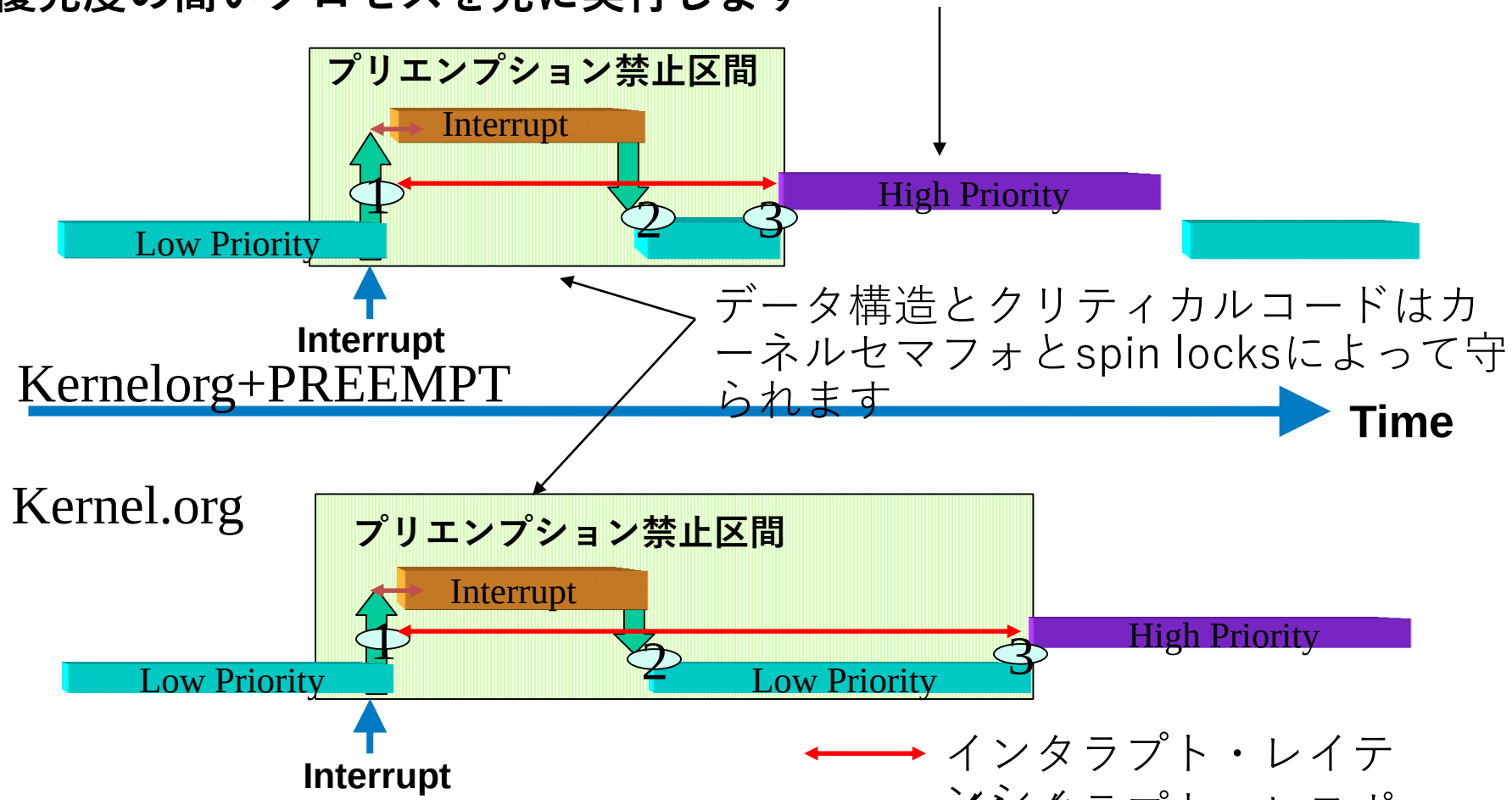
多重割込みとプリエンプション

- 最高優先度のプロセスよりハードウェア割込みが優先される
- 多重割り込み：割込みは、IOの数だけ発生する可能性がある
- 割込みの副作用を排除する方法
  - シングルコア・アプローチ
    - プリエンプション vs 割込み禁止区間
    - SoftIRQ,タスクレット,IRQデーモン→（割り込みと優先度タスクの調停）
    - IOの数を制限する→多重割込み時の応答時間の確定
  - マルチコア・アプローチ(RedHawk)
    - プリエンプション
    - CPUシールド（割込みを任意のCPUに割付けて処理させる）

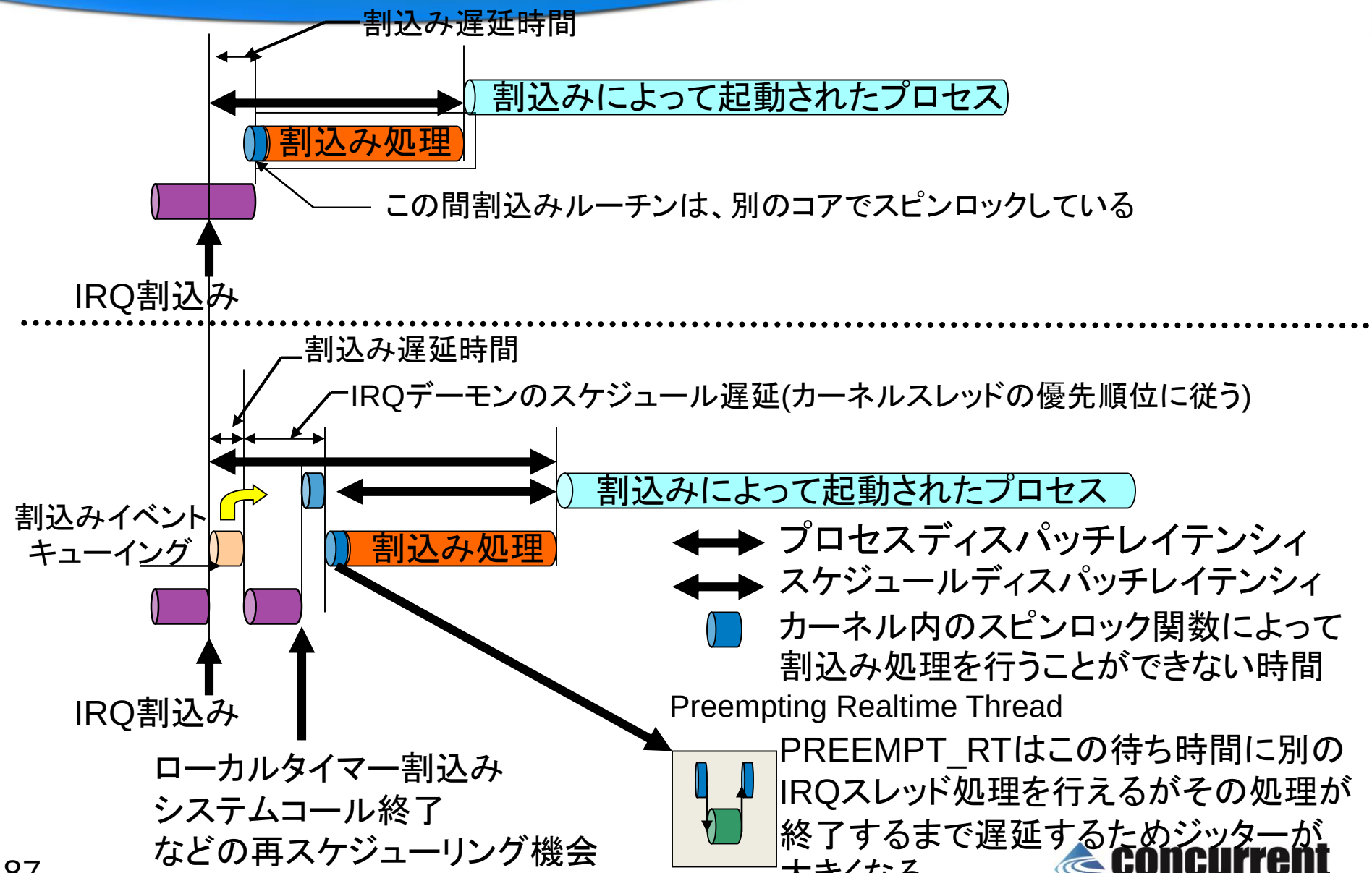


PREEMPTカーネルと標準カーネル

プリエンプション
優先度の低いプロセスがカーネル内を実行中でも
優先度の高いプロセスを先に実行します



PREEMPT vs PREEMPT_RT

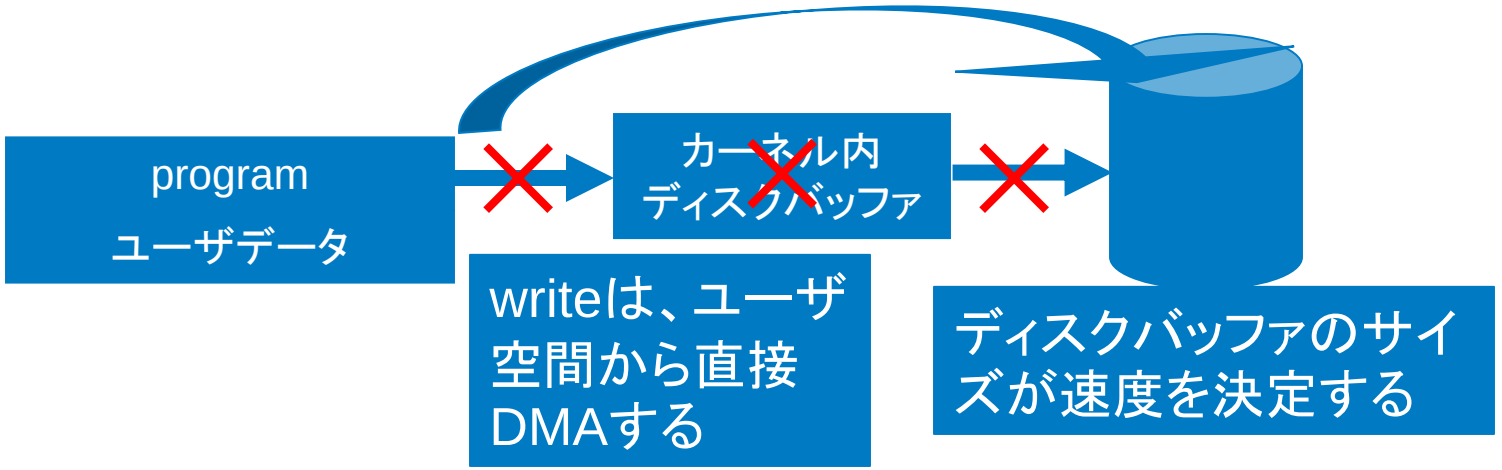


“sleeping spinlocks” スピンロック中のスリープ

- **“sleeping spinlocks”**は、PREEMPT_RT パッチの重要なモジュールです。
- 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 spin_lock_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、システムレスポンスに否定的な影響を与えることがあります。

RedHawk ダイレクトIOプログラム

- ダイレクトIOを使って、ユーザ空間から直接するDMA方法
  - バッファが大きいとスループットは向上しますが、絶対時間が間に合わないことがあります。
  - その場合にはバッファサイズを小さくすることで、書き出 し時間を調整するか、書き出しプロセスとデータ生成プロセスを分離して記述する必要があります。



concurrent REAL-TIME

非POSIX API

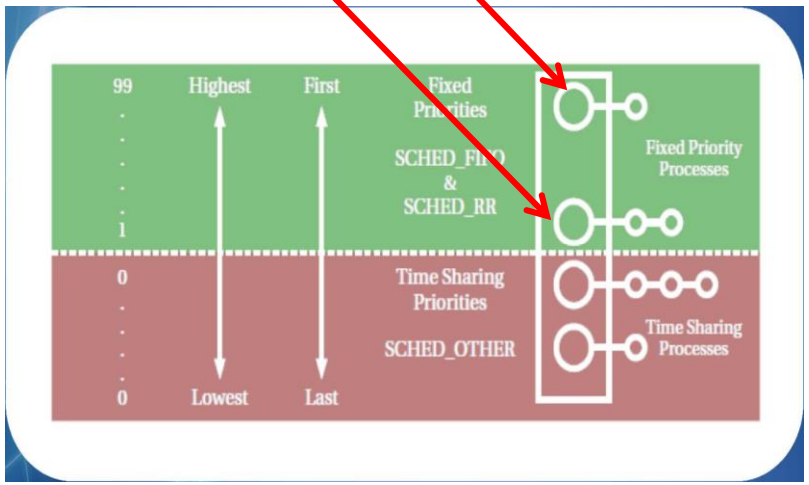
- CPU指定
  - Linux API
    sched_getaffinity(getpid(),cpusetsize,&mask);
    sched_setaffinity(getpid(),cpusetsize,&mask);
    cpuset_set_cpu(setp,cpu,1);/* 1:set */
  - RedHawk API
    mpadvise (MPA_CPU_PRESENT,0,0,setp);
    mpadvise (MPA_CPU_ACTIVE,0,0,setp);
    mpadvise (MPA_PRC_GETBIAS,MPA_PID,0,setp);
    mpadvise (MPA_PRC_GETRUN,MPA_PID,0,setp);

RedHawk リアルタイムクロックとリアルタイムタイマ

- nanosleep(&request, &remain); (Relative Time)
- clock_nanosleep(clock_id, flags, &request, &remain);
  - clock_id
    - CLOCK_REALTIME
      - A settable system-wide real-time clock.
    - CLOCK_MONOTONIC
      - A nonsettable, monotonically increasing clock that measures time since some unspecified point in the past that does not change after system startup.
    - CLOCK_PROCESS_CPUTIME_ID
      - A settable per-process clock that measures CPU time consumed by all threads in the process.
  - flags
    - TIMER_ABSTIME
- clock_gettime(clock_id, ,&curr_value);
- timer_id = timer_create(clock_id,&event,&timer) ;
- timer_settime(timer_id,flags, &new_value, &old_value) ;
- timer_gettime(timer_id,&curr_value);

実行スケジューリング

- 99:max = sched_get_priority_max(SCHED_FIFO);
- 0:min = sched_get_priority_min(SCHED_FIFO);
- 99:max = sched_get_priority_max(SCHED_RR);
- 0:min = sched_get_priority_min(SCHED_RR);



- myparam.sched_priority = 優先度;
- sched_setscheduler(getpid(),スケジューリング,&myparam);
- sched_rr_get_interval(getpid(),&rr);

RedHawk シグナル

- sigqueue(getppid(),SIGRTMAX,value);

  union sigval value;

  value.sival_int = 値;
- シグナルマスク
- sigaction()

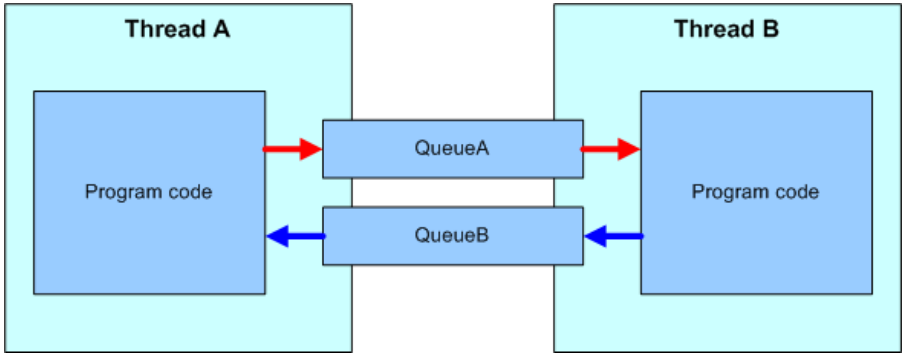
  SA_SIGINFO , SA_RESTART フラグ
- sigwait()
- sigsuspend()

RedHawk 共有メモリ

- oflg = O_CREAT | O_RDWR| O_TRUNC;
- omode = 0644;
- memfd = shm_open(“名前”, oflg, omode);
  - /dev/shm/名前 が作られる
- ftruncate(memfd, サイズ) ;
- shmaddr=mmap(0,サイズ,PROT_READ | PROT_WRITE,MAP_SHARED,memfd,0);
- munmap(shmaddr,サイズ) ;
- close(memfd);

メッセージキュー

- **comattr.mq_flags = O_NONBLOCK;**
- comattr.mq_maxmsg = 100;
- comattr.mq_msgsize = sizeof(int);
- comattr.mq_curmsgs = 0;
- fbq = mq_open("/fbq",O_RDWR|O_CREAT|**O_NONBLOCK**,0666, &comattr);
- mq_send(fbq,(char*)&send,sizeof(int),i) ;
- mq_receive(fbq,(char*)&recv,sizeof(int),&msg_prio);
- notification.sigev_notify = SIGEV_SIGNAL;
- notification.sigev_signo = signum;
- mq_notify(fbq,&notification);
- mq_close(fbq);
- mq_unlink("/fbq");



/proc インタフェース

POSIX メッセージキューが消費するカーネルメモリの量を制限することができる。

- **/proc/sys/fs/mqueue/msg_max**
  こ のファイルを使って、一つのキューに入れられるメッセージの最大数の上限値を参照したり変更したりできる。この値は、mq_open(3) に渡す attr->mq_maxmsg 引き数に対する上限値として機能する。msg_max のデフォルト値および最小値は 10 である; 上限は「埋め込みの固定値」(HARD_MAX) で (131072 / sizeof(void *)) (Linux/86 では **32768**) である。この上限 は特 権プロセス (CAP_SYS_RESOURCE) では無視されるが、埋め込みの固定値による上限はどんな場合にでも適用される。

- **/proc/sys/fs/mqueue/msgsize_max**
  このファイルを使って、メッセージの最大サイズの上限値を参照したり変更し たり でき る。この値は、mq_open(3)に渡すattr.mq_msgsize 引き数に対する上限値として機能する。msgsize_maxのデフォルト値および 最小値は 8192 バイトである; この上限は特権プロセス(CAP_SYS_RESOURCE)では無視される。

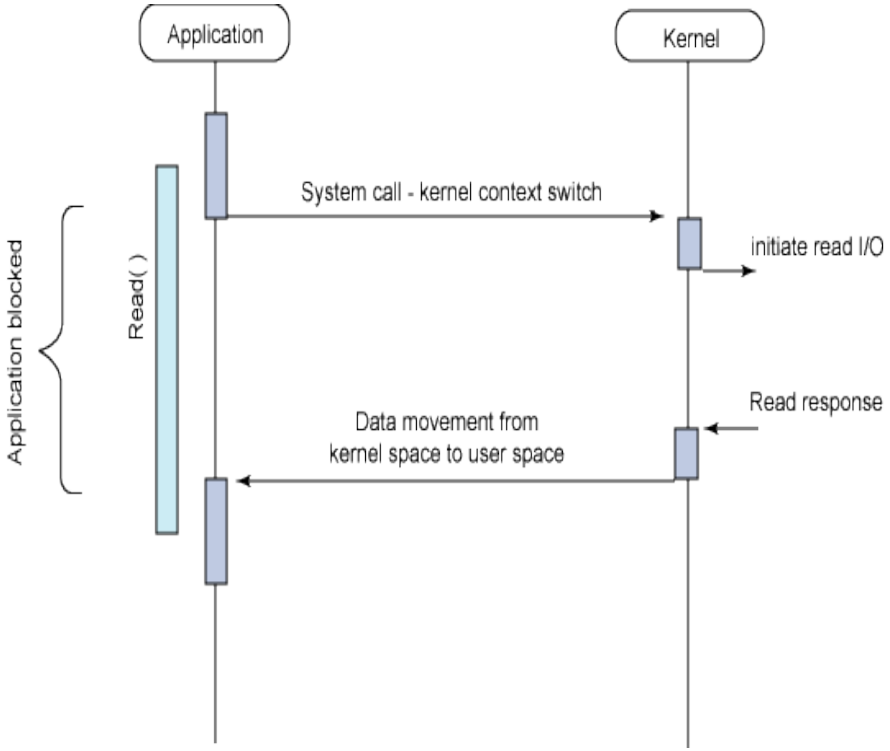
- **/proc/sys/fs/mqueue/queues_max**
  こ のファイルを使って、作成することができるメッセージキューの数に対するシステム全体での制限を参照したり変更したりできる。一度この上限に達すると、新しいメッセージキュー を作成できるのは特権プロセス(CAP_SYS_RESOURCE)だけとなる。queues_maxのデフォルト値は 256 であり、0 から INT_MAX の範囲の任意の値に変更することができる。

Linux IOモデル

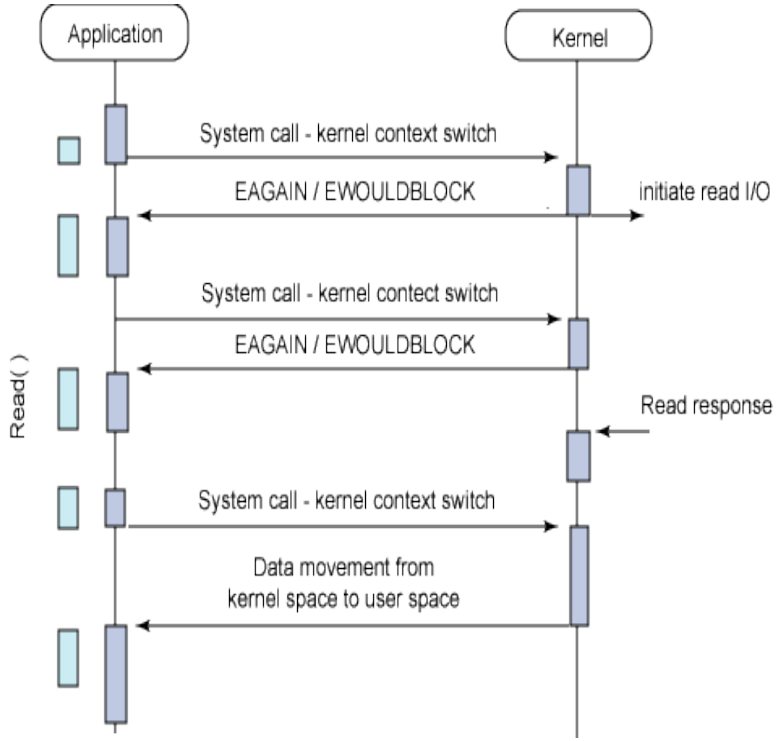
	ブロッキング	非ブロッキング
同期	**Synchronous blocking I/O** Read/Write 同期I/O	**Synchronous non-blocking I/O** Read/Write(O_NONBLOCK) ポーリングI/O
非同期	**Asynchronous blocking I/O** select/poll I/O マルチプレクシング	**Asynchronous non-blocking I/O (AIO)** 非同期 I/O （スレッドによる並列I/O）

同期IO

- **Synchronous blocking I/O**

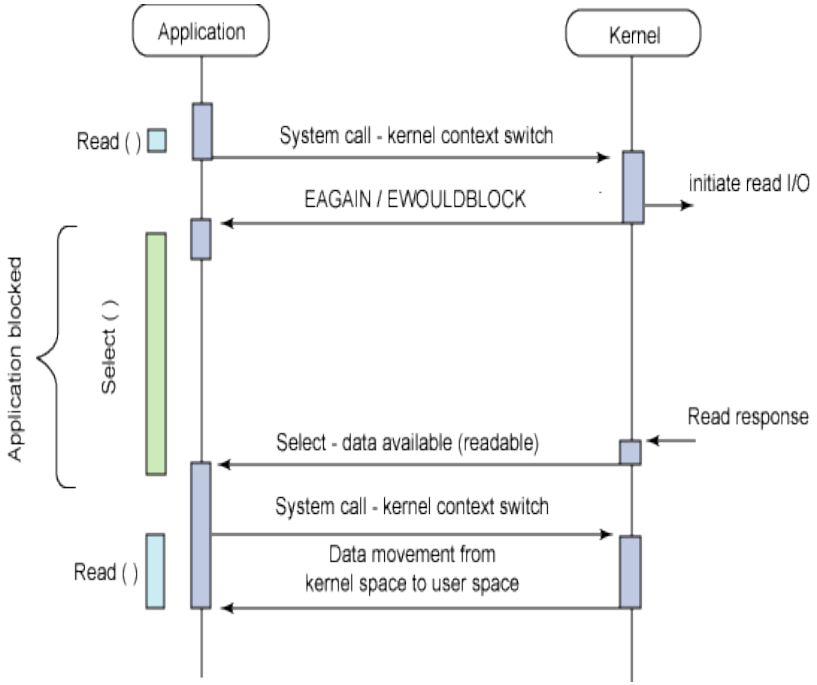


- **Synchronous non-blocking I/O**

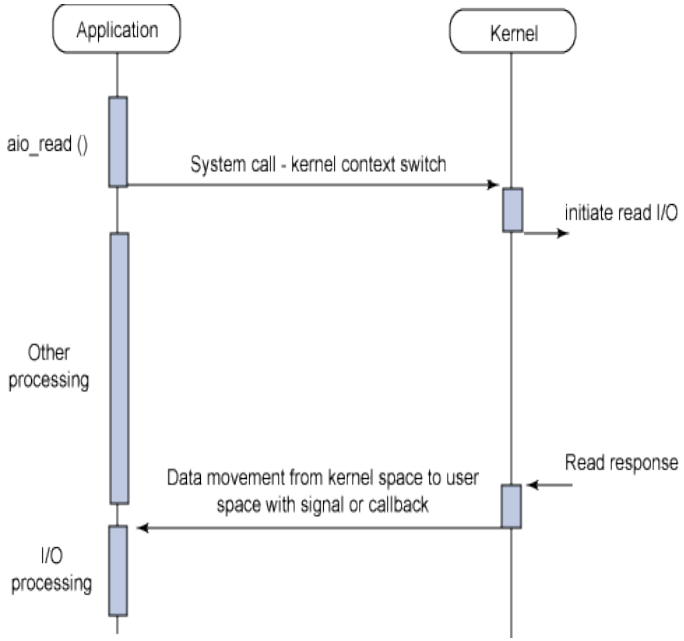


非同期 I/O

- **Asynchronous blocking I/O**



- **Asynchronous non-blocking I/O (AIO)**



同期 I/O

```
#define _LARGEFILE64_SOURCE
#include <stdio.h>
#define __USE_GNU
#include <sys/types.h>
#include <unistd.h>
#include <sys/stat.h>
#include <fcntl.h>
#include <stdlib.h>

main()
{
        int fd;
        off64_t align;
        long long bytes;
        char *dummy;

        bytes = 4294967296LL;//0x100000000;
        if ((fd = open64("realtime_file",(O_CREAT|O_WRONLY|O_DIRECT), 0666)) <0)
        {
                exit(1);
        }
        align = (off64_t)fpathconf(fd,_PC_ALLOC_SIZE_MIN);
        printf("_PC_ALLOC_SIZE_MIN %lld¥n",align);
        printf("allocate %lld bytes¥n",bytes);
        posix_memalign((void*)&dummy,align,0x1000);
        if (posix_fadvise(fd,0LL,bytes,POSIX_FADV_NOREUSE)<0)
        {
                fprintf(stderr,"fadvise error¥n");
                exit(0);
        } else {
                if (posix_fallocate64(fd,0LL,bytes)<0)
                {
                        fprintf(stderr, "falocate error¥n");
                        exit(0);
                }
                bytes -= 4096LL;
                if (lseek64(fd,bytes,SEEK_SET)<0)
                {
                        fprintf(stderr,"lseek error¥n");
                }
                write(fd,dummy,0x1000);
        }
        close(fd);
}
```

非同期 I/O

```
struct aiocb Aiocb[DATA_BUF_NUM];
struct aiocb *List[DATA_BUF_NUM];
struct timespec timeout = {10,0};
int fd; struct sigevent sig;
fd = open("realtime_file",(O_CREAT|O_DIRECT|O_WRONLY|O_SYNC|O_DSYNC|O_RSYNC), 0666) ;
for(n=0;n<DATA_BUF_NUM;n++)
 {
      Aio_buff[n] = (unsigned char*)memalign(sysconf(_SC_PAGESIZE),DATA_BUF_SIZE);
      Aiocb[n].aio_buf = Aio_buff[n];
      Aiocb[n].aio_offset = (long long)(DATA_BUF_SIZE * n);
      Aiocb[n].aio_nbytes      = (long long)DATA_BUF_SIZE;
      Aiocb[n].aio_fildes          = fd;
      Aiocb[n].aio_reqprio     = 0;
      Aiocb[n].aio_lio_opcode  = LIO_WRITE;
      Aiocb[n].aio_sigevent.sigev_notify = SIGEV_NONE;
      List[n] = &Aiocb[n];
}
lio_listio(LIO_WAIT,(struct aiocb **)&List[0],DATA_BUF_NUM, &sig);
aio_suspend((const struct aiocb **)&List[0],DATA_BUF_NUM, &timeout);
close(fd)
```

メモリロッキング

- align = (off64_t)fpathconf(fd,_PC_ALLOC_SIZE_MIN);
- **posix_memalign((void*)&dummy,align,0x1000);**
- **mlockall(MCL_CURRENT|MCL_FUTURE);**
- **munlockall();**

セマフォ

```
sem_t *sem;
int pshared=2;      /* 2 process shared */
int value=0;
int sval,status;
sem = sem_open("/posix_sem0",O_CREAT|O_EXCL,0644,0); //create /dev/shm/sem.posix_sem0
sem_init(sem,pshared,value);
sem_getvalue(sem,&sval); printf("initial semaphore value is %d¥n",sval);
sem_post(sem);
sem_getvalue(sem,&sval); printf("after sem_post semaphore value is %d¥n",sval);
sem_wait(sem);
sem_getvalue(sem,&sval); printf("after sem_wait semaphore value is %d¥n",sval);
status = sem_trywait(sem);
if (status)             printf("status %d %s¥n",status,strerror(errno));
else    printf("successful lock a counting semaphore¥n");
sem_getvalue(sem,&sval); printf("current semaphore value is %d¥n",sval);
sem_post(sem);
status = sem_trywait(sem);
if (status)             printf("status %d %s¥n",status,strerror(errno));
else    printf("successful lock a counting semaphore¥n");
sem_getvalue(sem,&sval); printf("current semaphore value is %d¥n",sval);
sem_close(sem);
sem_unlink("/posix_sem0");
```

pthreadのシグナルハンドリング

pthread_sigmask, pthread_kill, sigwait

スレッド内でのシグナルハンドリング

書式

```
#include <pthread.h>

#include <signal.h>

int pthread_sigmask(int how, const sigset_t  *newmask,  sigset_t  *old mask);

int pthread_kill(pthread_t thread, int signo);

int sigwait(const sigset_t *set, int *sig);
```

説明

pthread_sigmaskは呼び出しスレッドのシグナルマスクを引数 how および newmask で指定されるように変更する。oldmask が NULL でないときには、直前のシグナルマスクが oldmask で指し示される領域に格納される。

引 数 how および newmask の意味は sigprocmask(2) の引数の意味と同じである。how が SIG_SETMASK のときには、シグナルマスクが newmask に設定される。how が SIG_BLOCK のときには、newmask で指定されるシグナルが現時点のシグナルマスクに追加される。how が SIG_UNBLOCK のときには、 newmask で指定されるシグナルが現時点のシグナルマスクから取り除かれる。

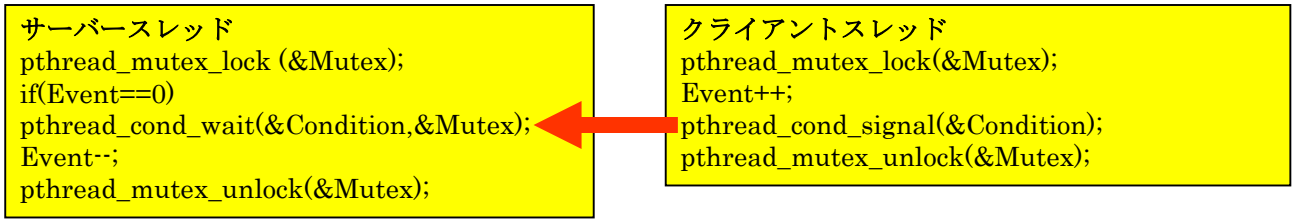
シグナルマスクはスレッドごとに設定されることを思い出してほしい。しかし sigaction(2) で設定されるシグナルアクションとシグナルハンドラは、すべてのスレッドで共通である。

pthread_kill はシグナル番号 signo のシグナルをスレッド thread に送信する。シグナルは kill(2) に書かれているように配送されハンドルされる。

sigwait は set で指定されるシグナルのうちいずれか 1 つが呼び出しスレ ッド に配送されるまで呼び出しスレッドの実行を停止する。そして受信したシグナルの数を sig で指し示される領域に格納して返る。set で指定されるシ グナルは sigwait に入るときにブロックされていなければならず、無視されてはならない。**配送されたシグナルに対するシグナルハンドラが登録されていてもハンドラ関数は呼び出されない。**

PTHREAD間の同期

- pthread_mutex_lock()/pthread_mutex_unlock()は、mutex変数 Mutexで、クリティカルセクションを作っています。
- この例でのクリティカルセクションデータはEventで、この値を操作する間、Mutexで保護されています。今、サーバスレッド側で、Evantの値が0である（つまりイベントが無い）とき、スレッドは、pthread_mutex_cond_wait()で、Mutexを解放し、Condition変数で、眠りにつきます。クライアントスレッド側は、スレッドがMutexを解放している間に、Eventを加算し、pthread_cond_signal()を発行します。
- 休眠状態にあった、スレッドは、クライアントスレッド側がMutexを解放した時に、pthread_cond_wait()から目覚め、そのとき再びMutexをロックします。
- この動作で、Mutex変数Eventはイベントカウンタとして動作するため、イベントを失うことなく、スレッドの実行ー停止を繰り返すことができます。



ご質問は？