Tasklet 方式による Linux のリアルタイム性向上

齋藤真輝、阿部昌裕、片桐敦（Ａ＆Ｄ）
曽我正信、中出実、山中禎詠（三菱重工業）
杉岡利信（ＩＴＯ）、小島一元（フリープログラマ）
比屋根一雄、飯尾淳、谷田部智之（三菱総研）

Copyright(C) 2004 r2linux developer team

発表内容

- なぜ、リアルタイム性向上の改良が必要か？
- 通常カーネルのリアルタイム性
  - Low Latency + Preemptionパッチ適用後も遅れる
- ドライバのTasklet化による起動遅れ改善方法
  - IDEドライバ、Ethernetドライバにおける実例
- リアルタイム性向上結果
- その他の改良点

Copyright(C) 2004 r2linux developer team

なぜ、リアルタイム性向上の改良が必要か

- 組込み機器では一定周期で確実に動作が必要
  - リアルタイムタスクの起動遅れは致命的
- リアルタイム性の阻害要因は様々
  - アプリケーション起動／終了
  - ファイル入出力
  - ネットワーク通信
  - メモリ確保／解放
- しかし、Linux-2.4.20 ではリアルタイム性が不足
  - まずは、測ってみよう

Copyright(C) 2004 r2linux developer team

ターゲット環境

- SH4 プロセッサ搭載の５種類の評価ボード

製造元	製品名	CPU	CPU clock
オムロン	QT-PIDS	SH7750R	236MHz
アドバネット	FXCPU01	SH7751R	240MHz
日立超*LSI*	MS7751RSE01	SH7751R	240MHz
ＣＱ出版	SH4-PCI with Linux	SH7751	167MHz
京都マイクロ	Solution Platform for SH4R	SH7751R	240MHz

- ホスト PC から RS232C と LAN で接続
  - RS232C 経由でログインし、モニタリング

Copyright(C) 2004 r2linux developer team

リアルタイム性の測定方法

- リアルタイムタスクを 10msec 毎に起動し、５種類の負荷をかけた状態で、起動遅れを計測する
  - タイマ割込→ ISR 起動→ドライバ起動→アプリ起動
- 負荷
  - (1) アプリケーション起動／終了
    - bash の起動と終了を無限ループで連続実行
  - (2) ファイル I ／ O
    - ファイルのコピーと削除を無限ループで連続実行
  - (3)Ftp get ／ (4)Ftp put
    - ホスト PC で ftp コマンドでファイル (10MB x 10 個) を SH4 評価ボードから ftp 受信／ ftp 送信を無限ループで連続実行
  - (5) メモリ確保／解放
    - Malloc(), memset(), free() を無限ループで連続実行

Copyright(C) 2004 r2linux developer team

測定例

- 測定プログラム period_hist の動作例

Commands | Programs | Terminal | shell

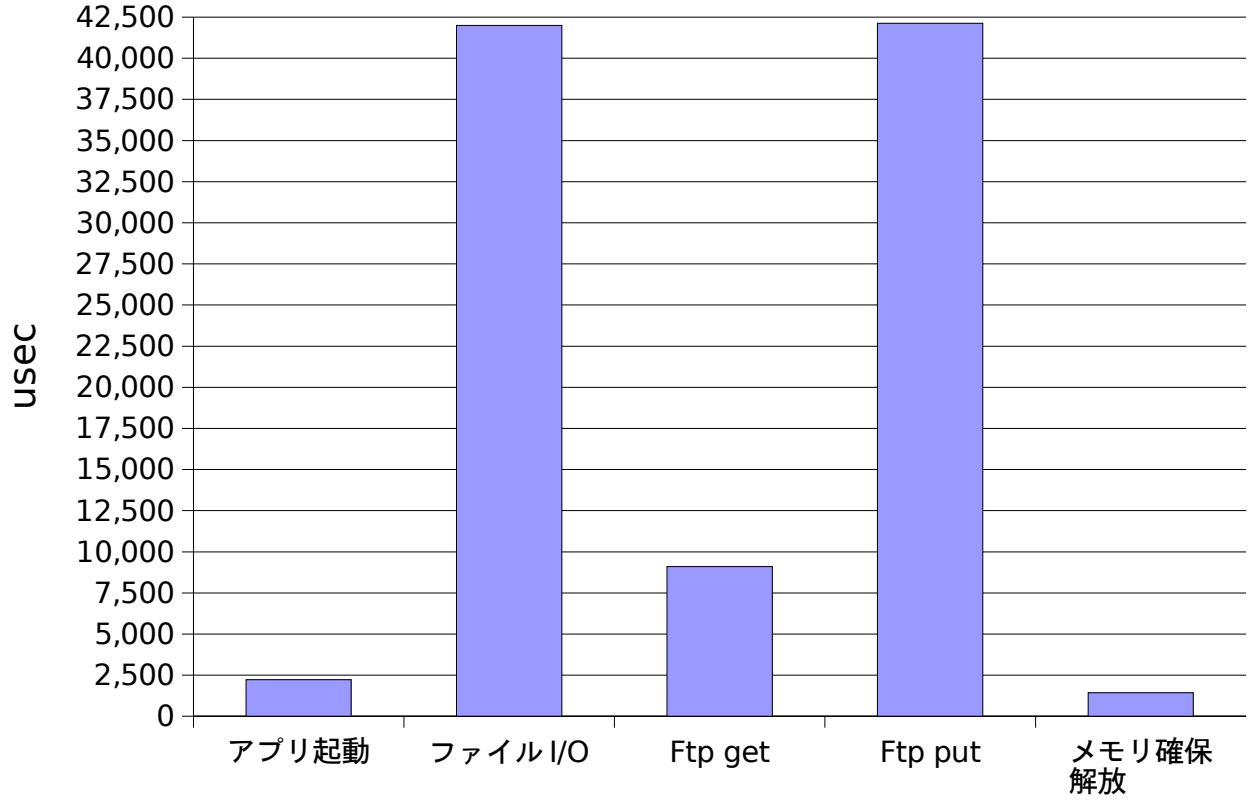
```
tyatabe:kterm@evah{/tmp }[379]%./period_hist
dist_threshold[usec], 499
task period[usec], 10000
                                  Max -App Wake-up Interval----- -App Int.      Since Start- [usec]
date/time              Delay,   Delay,     Min      Max   Jitter,     Min       Max   Jitter
-------------------=======-=======-=======-=======-=======-=======-=======-=======
2004/05/26 04:49:21,      61,      61,    9970,   10015,       29,    9970,    10015,       29,
2004/05/26 04:49:22,      65,      65,    9978,   10035,       35,    9970,    10035,       35,
2004/05/26 04:49:23,      65,      65,    9981,   10034,       34,    9970,    10035,       35,
2004/05/26 04:49:24,      57,      65,    9982,   10026,       26,    9970,    10035,       35,
```


```
                                  Max -App Wake-up Interval----- -App Int.      Since Start- [usec]
date/time              Delay,   Delay,     Min      Max   Jitter,     Min       Max   Jitter
```


Copyright(C) 2004 r2linux developer team

測定結果（１）

- リアルタイム周期 10msec に対し、最大 40ms 以上の起動遅れ
  - 特に、ファイル I／O と Ftp put が遅い
  - 目標 500usec に対しては、すべて大きく未達



評価ボード
オムロン QT-PIDS

Copyright(C) 2004 r2linux developer team

Low Latency Patch と Preemption Patch

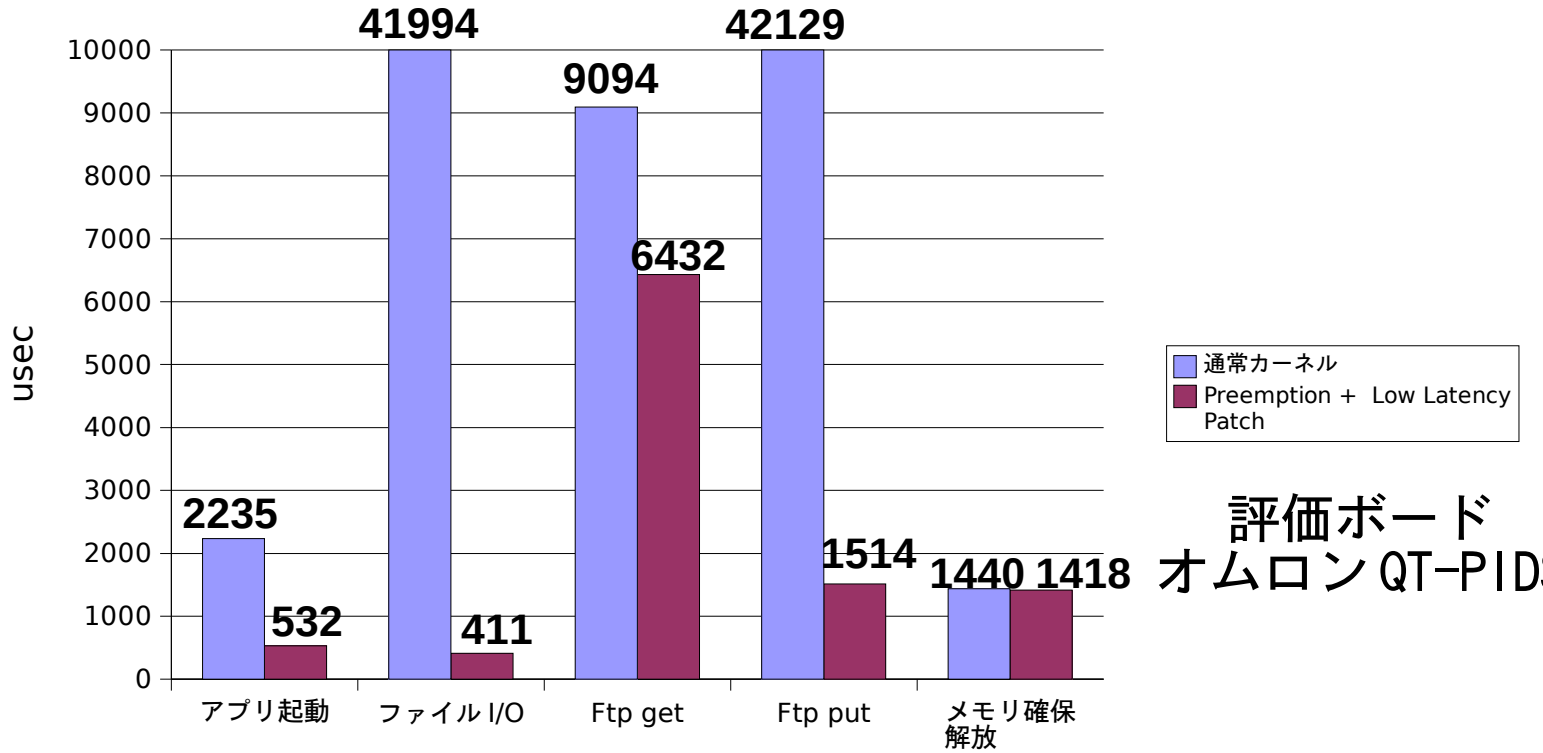
- Low Latency Patch
  - カーネルやデバイスドライバ中に存在するタスク切替え可能箇所を試行錯誤的に見つけ、タスク切替え処理を追加する
- Preemption Patch
  - 応答速度向上のためにカーネル空間実行中に割込みが発生した場合にも、割込み終了時にタスクの切替え（プリエンプション）を可能にする

- とりあえず当ててみる
  - うまく当らない所もあるが成功

Copyright(C) 2004 r2linux developer team

測定結果（２）

- Low Latency Patch + Preemption Patch
  - アプリ起動、ファイル I/O 、 Ftp put は画期的に改善
  - Ftp get 、メモリ確保／解放はあまり改善しない
  - 目標 500usec 以内は、ファイル I/O のみ



評価ボード
オムロン QT-PIDS

Copyright(C) 2004 r2linux developer team

Patchの現状と対策案

- Low Latency Patch
  - 性能限界がはっきりしない
  - デバイスによっては全くパッチをあてていない。
  - （対策）新たなタスク切替え処理を追加
- Preemption Patch
  - 現時点では性能が充分ではない
  - タスク切替えできない３つのケース
    - （１）割込み禁止
    - （２）スピンロック（spin_lock()）
    - （３）ソフト割込み禁止（local_bh_disable()）
  - （対策）長い禁止箇所を見つけ分割する

Copyright(C) 2004 r2linux developer team

イベントログによる遅れの原因究明

- タスク切替え遅延箇所を「イベントログ機能」で探す
  - イベント発生時刻記録ツールを開発
  - ログはメモリ内に保持し、低負荷で記録できる
  - 遅延が設定値以上の場合、ログを停止する

取得できるイベント種類

タスク切替え、シグナル発生
LowLatencyPatch、PreemptionPatchによるタスク切替え箇所
BottomHalf、例外、割込、システムコール開始／終了
ファイル読込み／書出しの開始／終了
スピンロック、割込禁止の開始／終了等

Copyright(C) 2004 r2linux developer team

イベントログの例（Ftp get）

```
タイマカウント 経過時間(μ秒)     イベント
78BF84B0      -6463      Inter:TimerUnit1 PR=88104C4EH SPC=8800D0E0←タイマー1割込
78BF8482      -6458        irq_enter:  flag=20000000H PC=88009E08H
78BF8480      -6458        ISR ENTER, PC=C01591A0H
78BF8460      -6454        timer1 interrupt : delta_time=10005←タスク起動命令
78BF840C      -6446        ISR EXIT,  PC=C01591A0H
78BF8405      -6445        irq_exit :  local_irq_count=1 PC=88009E5CH
78BF83E8      -6442      restore_all:  PR=88104C4EH SPC=8800D0E0H←タイマー1割込終了
78BF81A7      -6383      Inter:IRQD PR=88104CD0H SPC=88104CDC←Ethernet割込
78BF8190      -6381        irq_enter:  flag=00000000H PC=88009E08H
78BF8188      -6380        ISR ENTER, PC=88105360H
78BF8142      -6373        ISR EXIT,  PC=88105360H
78BF813B      -6372        irq_exit :  local_irq_count=1 PC=88009E5CH
78BF8125      -6370      restore_all:  PR=88104CD0H SPC=88104CDCH←Ethernet割込終了
78BF8102      -6367      Inter:IRQC PR=8811AD22H SPC=8811AD2E←IDE割込
78BF80EF      -6365        irq_enter:  flag=20000000H PC=88009E08H
78BF80EA      -6364        ISR ENTER, PC=8810AC80H
78BF7FF5      -6339      Inter:TimerUnit0 PR=8800FD6EH SPC=8800FD9C←OSタイマー割込
78BF7FE0      -6337        irq_enter:  flag=20000000H PC=88009E08H
78BF7FDD      -6337        ISR ENTER, PC=8800BF00H
78BF7FD4      -6336        timer0 interrupt : delta_time=10001usec
78BF7F6A      -6325        ISR EXIT,  PC=8800BF00H
78BF7F64      -6325        irq_exit :  local_irq_count=2 PC=88009E5CH
78BF7F46      -6322      restore_all:  PR=8800FD6EH SPC=8800FD9CH←OSタイマー割込終了
78BF7CF6      -6261        ISR EXIT,  PC=8810AC80H
78BF7CF1      -6261        irq_exit :  local_irq_count=1 PC=88009E5CH
78BF7CDD      -6259      restore_all:  PR=8811AD22H SPC=8811AD2EH←IDE割込終了
78BF7C7A      -6249      Inter:IRQC PR=88122626H SPC=8812254E←IDE割込
                                    (IDE割込中)
78BF797D      -6171      restore_all:  PR=88122626H SPC=8812254EH←IDE割込終了
                                       :
```

Copyright(C) 2004 r2linux developer team

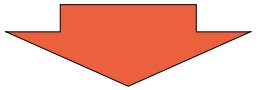
イベントログの例（Ftp get）（つづき）

```
タイマカウント 経過時間(μ秒)      イベント
78BF782B      -6136       Inter:IRQC PR=8800BD66H SPC=8800BD72←IDE割込
                                  (IDE割込中)
78BF74DA      -6050       restore_all:  PR=8800BD66H SPC=8800BD72H←IDE割込終了
                                  :
78BF73E5      -6025       Inter:IRQC PR=88104C4EH SPC=8800FFAE←IDE割込
                                  (IDE割込中)
78BF70E8      -5947       restore_all:  PR=88104C4EH SPC=8800FFAEH←IDE割込終了
78BF6F50      -5906       Inter:IRQD PR=88104CD0H SPC=88104CDC←Ethernet割込
                                (Ethernet割込中)
78BF6EE8      -5895       restore_all:  PR=88104CD0H SPC=88104CDCH←Ethernet割込終了
                                  :
  ■IDE割込          合計  ３７回  ３０３６μsec  起動妨害
  ■Ethernet割込  合計  ２８回  ４４０μsec  起動妨害
                                  :
78BE9BB2       -395         local_bh_enable() : 2->1 PC=8812256CH
78BE9B17       -379         local_bh_disable() : 1->2 PC=8813F954H
78BE9B07       -378         local_bh_enable() : 2->1 PC=8813F992H
                                  :
78BE8F12        -66         softirq action EXIT,  PC=88123280H
78BE8F01        -65         __local_bh_enable() : 1->0 PC=8801E612H
78BE8EDD        -61         preempt_schedule() : preempt_count=0 PC=880131B6H
78BE8ED0        -60         schedule() spin-cli=0H PC=880131D0H
78BE8E89        -52    0 in.ftpd (   67027 cnt) ->  49 ./period_hist   ←タスク切替
78BE8E23        -42       Excep:TLB Data Write Miss PR=C01592CAH SPC=88011A44
78BE8DE2        -35       restore_all:  PR=C01592CAH SPC=88011A44H
78BE8DA6        -29       restore_all:  PR=295852AEH SPC=2966B3C4H
78BE8D96        -28       Excep:TLB Code/Data Read Miss PR=295852AEH SPC=295852AE
78BE8D85        -26       restore_all:  PR=295852AEH SPC=295852AEH
                                  :
78BE8C8C         -1       trapa=17 sys_eventlogtimer1app() SPC=0040210AH
78BE8C82          0         timer1 -> app 0 : delta_time=6454   ←タスク起動確認
  注）割り込み禁止、スピンロックは非表示設定中
```

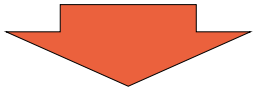
Copyright(C) 2004 r2linux developer team

タスク切り替えの遅延原因は割り込みそのもの

- 短い割り込みが何度も入ることが原因
- Patch の改良だけでは割り込みをコントロールできないため改善できない



- 割り込み処理＝ハード割り込み処理＋ソフト割り込み処理



- ハード割り込み処理の大部分をソフト割り込みへ移行（tasklet 化）すれば、割り込みをコントロールできる

- リアルタイムタスク実行中に割り込みが発生した場合、ソフト割り込みを保留後、再スケジュールし、ソフト割り込み完了までハード割り込みを禁止する

- リアルタイムタスク→非リアルタイムタスク切替え時に保留したソフト割り込みを実行する

Copyright(C) 2004 r2linux developer team

IDE ドライバの tasklet 化

- 問題点
  - IDE 割込み処理が他の処理をブロック
  - 割込み中にデータ転送するため時間がかかる
- 対策
  - IDE ドライバを tasklet 化
  - リアルタイムタスク実行中は、 tasklet を保留

Copyright(C) 2004 r2linux developer team

IDE ドライバの tasklet 化の実際

```
+++drivers/ide/ide.c
```

従来の割込みハンドラをリネーム

```
-void ide_intr (int irq, void *dev_id,
-               struct pt_regs *regs)
+static void ide_handle_intr(int irq,
+                            void  *dev_id)
 {
  unsigned long flags;
  ide_hwgroup_t *hwgroup
              = (ide_hwgroup_t *)dev_id;
  ide_hwif_t *hwif;
  ide_drive_t *drive;
  ide_handler_t *handler;
  ide_startstop_t startstop;
   (・・・省略・・・)
 }
```

tasklet 関数を作成

```
+#ifdef CONFIG_IDE_TASKLET
+static void ide_do_tasklet(unsigned long
+                           dev_id)
+{
+  ide_hwgroup_t *hwgroup
+              = (ide_hwgroup_t *)dev_id;
+  ide_hwif_t *hwif = hwgroup->hwif;
+  int irq = hwif->irq;
+
+  if (need_hi_tasklet_sched_ide()) {
+    ide_tasklet[hwif->index].data = dev_id;
+    tasklet_hi_schedule(&ide_tasklet
+                        [hwif->index]);
```

リアルタイムタスク実行中ならば
再スケジューリングして終了

```
+    return;
+  }
+
+  ide_handle_intr(irq, hwgroup);
+  enable_irq(irq);
+}
```

従来の割込みハンドラを呼び出す

割込み許可

新しい割込みハンドラを作成

```
+void ide_intr (int irq, void *dev_id,
+               struct pt_regs *regs)
+{
+  ide_hwgroup_t *hwgroup
+              = (ide_hwgroup_t *)dev_id;
+  ide_hwif_t *hwif = hwgroup->hwif;
+
+  disable_irq(irq);
+  ide_tasklet[hwif->index].data
+              = (unsigned long)dev_id;
+  tasklet_hi_schedule(&ide_tasklet
+                      [hwif->index]);
+}
```

割込み禁止

tasklet 実行

```
+#else
+void ide_intr (int irq, void *dev_id,
+               struct pt_regs *regs)
+{
+  ide_handle_intr(irq, dev_id);
+}
+#endif
```

Copyright(C) 2004 r2linux developer team

Ethernet ドライバの tasklet 化

- 問題点
  - 割込み処理の中で受信パケットを転送するのに時間がかかる
  - Ethenet 割込みは 1 回 100usec 以下であるが、何回か連続して入ることがある
- 対策
  - Ethernet ドライバを tasklet 化
  - リアルタイムタスク実行中は、 tasklet を保留
  - だが、長い tasklet のため、 tasklet 実行中にリアルタイムタスクの起動条件が整った場合、起動遅れが長くなる。（tasklet 中は local_bh_disable 状態のため）

  - tasklet 内部で、時々リアルタイムタスクの起動条件をチェックし、起動可能な場合は処理を中止し再スケジュール

Copyright(C) 2004 r2linux developer team

Ethernet ドライバの tasklet 化の実際

```
+++drivers/net/smc91111
-static void smc_interrupt
-                    (int irq, void * dev_id,
-                      struct pt_regs * regs)
+static void smc_handle_interrupt
+                    (int irq, void * dev_id)
 {
  struct net_device *dev   = dev_id;
  int ioaddr     = dev->base_addr;
  struct smc_local *lp
             = (struct smc_local *)dev->priv;
   (・・・省略・・・)

  /* set a timeout value,
     so I don't stay here forever */
  timeout = 4;

  PRINTK2(KERN_WARNING
      "%s: MASK IS %x \n", dev->name, mask);
  do {
+#ifdef CONFIG_NET_TASKLET
+    if (need_hi_tasklet_sched_net())
+      break;
+#endif
    /* read the status flag, and mask it */
    status = inb( ioaddr + INT_REG ) & mask;
    if (!status )
      break;
   (・・・省略・・・)
 }
```

従来の割込みハンドラをリネーム

リアルタイムタスク実行中ならば処理を終了

```
+#ifdef CONFIG_NET_TASKLET
+static void smc_do_tasklet
+                    (unsigned long dev_id)
+{
+  struct net_device *dev
+        = (struct net_device *)dev_id;
+  struct smc_local *lp
+        = (struct smc_local *)dev->priv;
+  int irq = dev->irq;
+
+  if (need_hi_tasklet_sched_net()) {
+    tasklet_hi_schedule(&lp->smc_tasklet);
+    return;
+  }
+
+  smc_handle_interrupt(irq, dev);
+
+  if (need_hi_tasklet_sched_net())
+    tasklet_hi_schedule(&lp->smc_tasklet);
+  else
+    enable_irq(irq);
+}
+
+static void smc_interrupt(int irq,
+ void * dev_id,  struct pt_regs * regs)
+{
+  struct net_device *dev = dev_id;
+  struct smc_local *lp
+            = (struct smc_local *)dev->priv;
+
+  disable_irq(irq);
+  tasklet_hi_schedule(&lp->smc_tasklet);
+}
```

tasklet 関数を作成

リアルタイムタスク実行中ならば再スケジューリングして終了

従来の割込みハンドラを呼び出す

再スケジューリング

割込み許可

新しい割込みハンドラを作成

割込み禁止

tasklet 実行

Copyright(C) 2004 r2linux developer team

デバイスドライバの tasklet 化の方法

- デバイス毎に tasklet 構造体を作成
  - デバイス構造体に tasklet 構造体を追加
  - デバイス構造体の初期化時に tasklet 構造体も初期化
    - tasklet_init() の第 3 引数にデバイス構造体のアドレスを渡す
- 従来の割込みハンドラをリネーム
  - xxx_intr()→xxx_handle_intr()
- 新しい割込みハンドラ xxx_intr() を作成
  - disable_irq() を呼出し、割込みを禁止
  - Tasklet 実行のため tasklet_hi_schedule() をコール
- Tasklet 関数 xxx_do_tasklet() の作成
  - リアルタイムタスク実行中ならば再スケジューリングして終了
  - 従来の割込みハンドラを呼び出す
  - enable_irq() を呼出し、割込みを許可

Copyright(C) 2004 r2linux developer team

リアルタイム優先度に応じたソフト割込みのマスク

- リアルタイムタスク実行中は tasklet 保留

↓

- リアルタイムタスクで通信・ファイル入出力したい

↓

- 所定のリアルタイム優先度（デフォルト 90）以上のリアルタイムタスク実行中は NET の tasklet 保留
- 所定のリアルタイム優先度（デフォルト 10）以上のリアルタイムタスク実行中は IDE の tasklet 保留

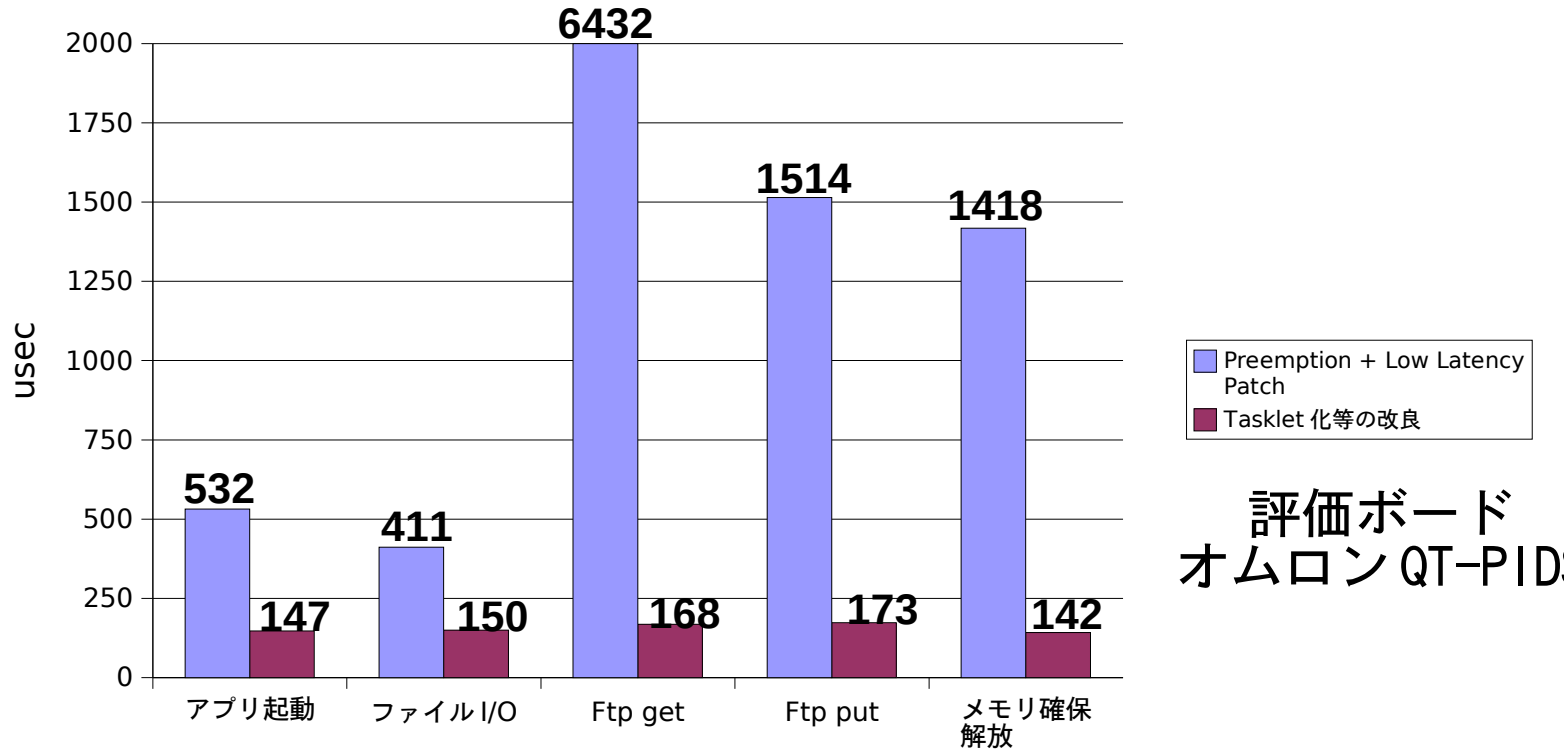
  - cf．リアルタイム優先度は１～９９

    通常の Linux プロセスのリアルタイム優先度は 0

Copyright(C) 2004 r2linux developer team

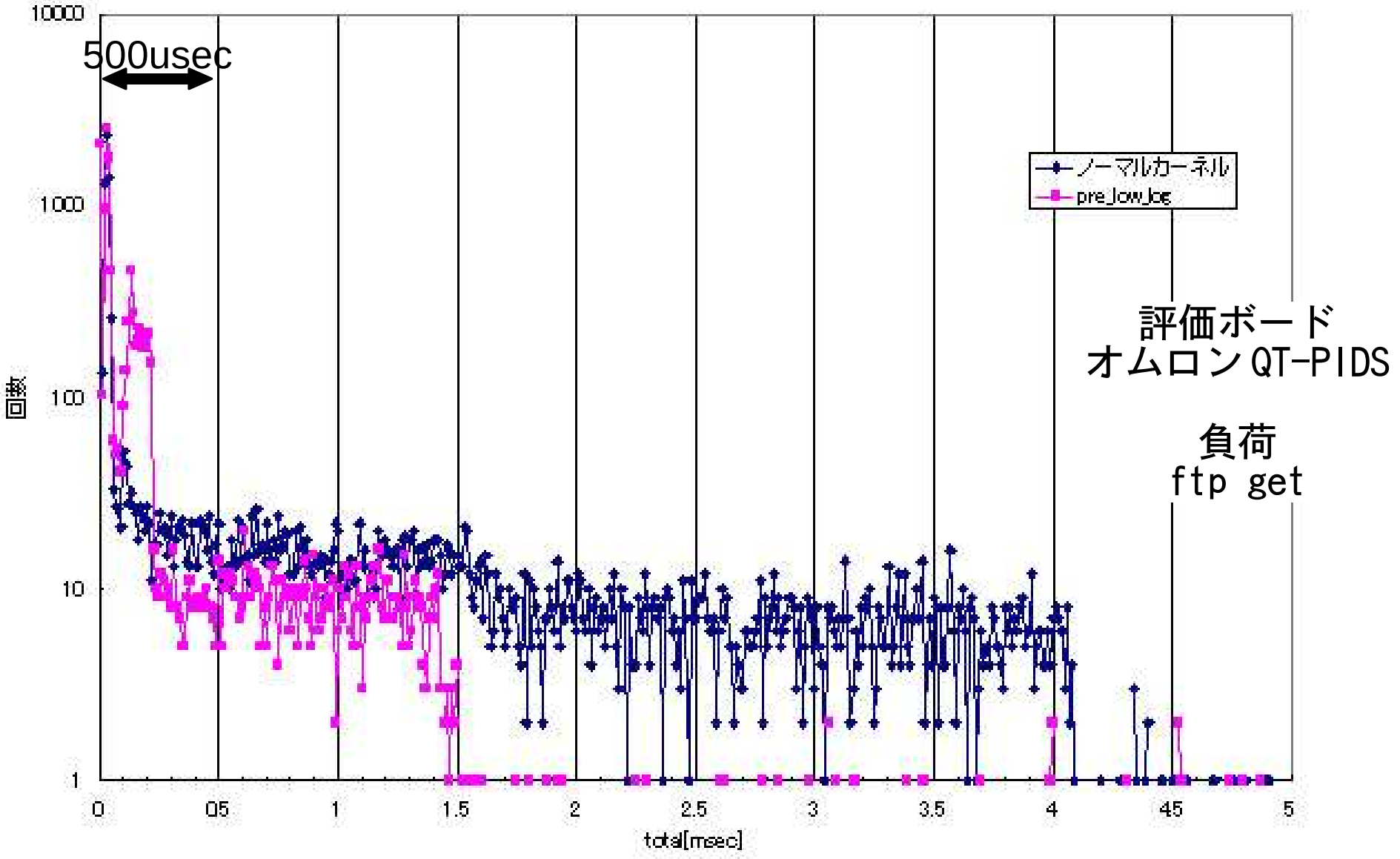
Tasklet 化の効果

- すべての負荷で最大起動遅れ 200usec 以下を実現
  - QT-PIDS 以外の評価ボードも 500usec 以下を実現



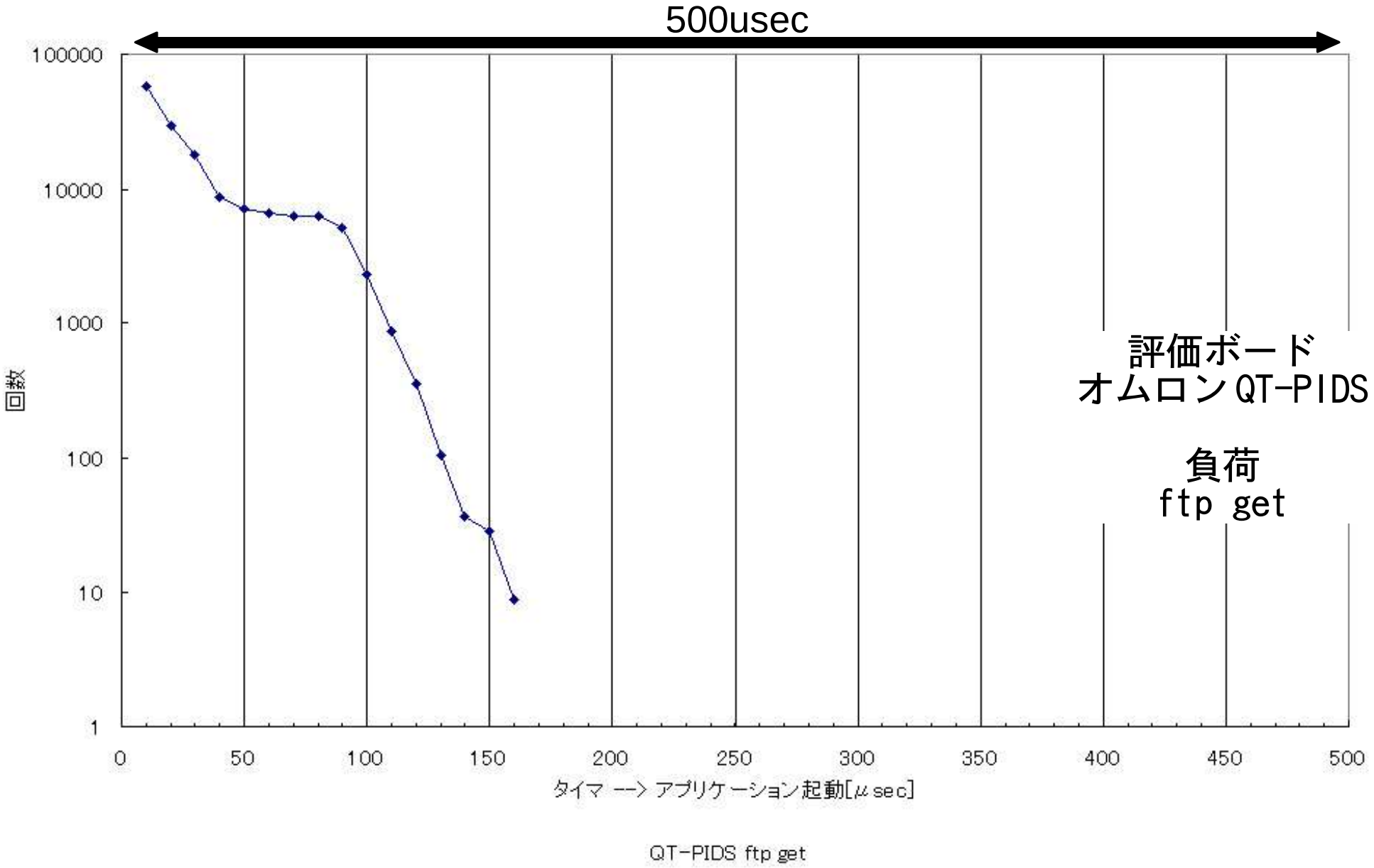
Copyright(C) 2004 r2linux developer team

起動遅れヒストグラム
（通常カーネル、LowLatency+Preemption Patch）



Copyright(C) 2004 r2linux developer team

起動遅れヒストグラム（Tasklet 化）



Copyright(C) 2004 r2linux developer team

その他の改良点

- ロックの分割
  - spin_lock(), local_bh_disable()の区間ではタスク切替えできない
    - Low Latency Patchと同様にプリエンプションポイントを挿入
- ins/outsの高速化（IDEドライバ内）
  - ループ展開してブロック期間を短縮
  - Ftp get起動遅れが、最大4.1msec→1.6msecに短縮
    （Tasklet化適用前の測定結果）
- printk()のデーモン化
  - 割込み処理からprintk()でエラーメッセージを出力するとポーリングで文字出力するため、リアルタイム性阻害
  - 文字列出力をkprintkdデーモンで処理するようにした

Copyright(C) 2004 r2linux developer team

まとめ

- デバイスドライバの tasklet 化によるリアルタイム性向上（起動遅れ改善）方法を開発した
- 数十 msec の最大起動遅れを、 200usec まで短縮
- Mission Critical な制御機器にも応用可能性が広がった
- SH-Linux カーネルにフィードバックし普及を図りたい
- 手法自体は SH-Linux に依存しないので、多くの Linux ドライバに適用可能な方法である

- 本開発の成果は以下で公開されている
  - http://r2linux.sourceforge.jp

本開発は独立行政法人情報処理推進機構（IPA）の
「平成 15 年度 オープンソフトウェア活用基盤整備事業」に採択された
「MissionCritical システム向け組み込み型リアルタイム Linux の開発」
の一部として開発された。

Copyright(C) 2004 r2linux developer team