スレッドの応答時間に関する実験
- 中間報告 -

松原 克弥
株式会社イーゲル
ルネサスソリューションズ

2005/9/30

背景

- ユーザレベルでデバイスドライバを実現したい
  - 開発が容易
  - ドライバのバグによるシステムダウンを軽減
  - (GPLの回避)
- ユーザレベル・デバドラ実現のためのいくつかの問題
  - I/Oメモリ、物理メモリへのアクセス
  - 割り込み要求 (IRQ) の転送
  - **割り込みへの応答速度**
  - ・・・・
- カーネル2.6の新機能
  - NPTL(Native POSIX Thread Library)
  - スケジューラの改善 (O(1)スケジューラ等)
  - カーネルプリエンプション
  - ・・・・

実験の目的

- スレッドの応答速度を測定
  - 特に、起床するときの時間
- RTスレッドの有効性を確認

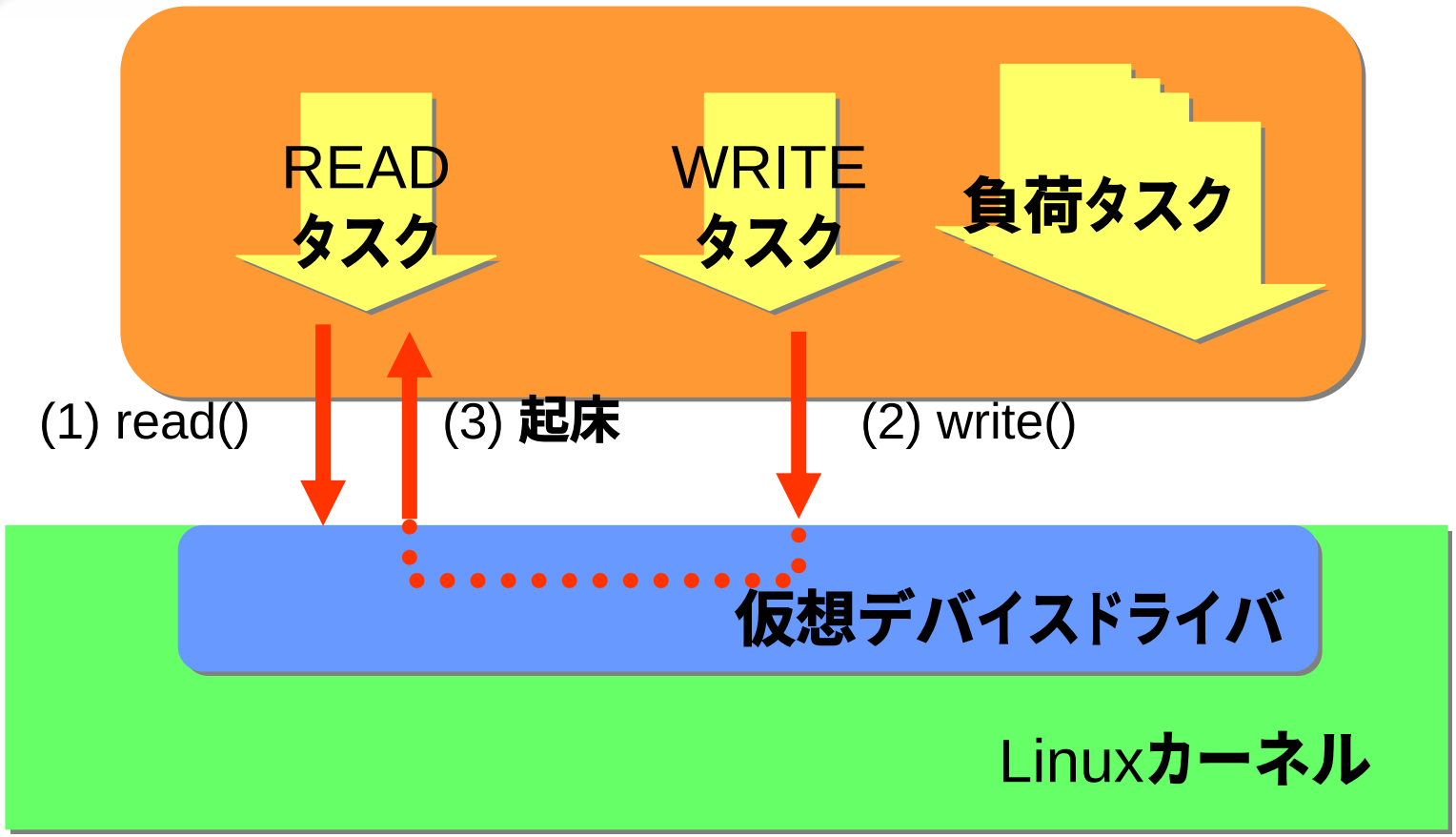
実験環境

- Renesas RTS7751R2D (big-endian) solution engine
  - SH4(SH7751) 240MHz
  - 64MB SDRAM
  - 10base-T Ethernet
- Linux 2.6.8.1
- Glibc 2.3.4
- RootfsはNFSサーバ上

実験プログラム構成

- 仮想デバイスドライバ
  - 仮想デバイスファイル (/dev/irqhook) を作成し、デバイスファイルへの書き込み (write) されたときに、同デバイスファイルに対してreadでブロックしているスレッドを起床する。
- WRITEタスク
  - 仮想デバイスファイルに対してwriteを発行することで、READスレッドの起床に対するトリガーを発行する
- READタスク
  - 仮想デバイスファイルに対してread()を発行し、ブロックする。
- 負荷タスク
  - 上記タスクと同じくスケジュール対象となるライバルタスク

実験プログラム構成 (contd.)



READタスク

```
void read_task () {
    ….

    /* 仮想デバイスファイルを開く */
    fd = open(“/dev/irqhook”, O_RDONLY);

    for(i=0; i<10000; i++) {
        /* read()を発行してブロック */
        read(fd, &tv1[i], sizeof(struct timeval));
        /* 時刻を記録 */
        gettimeofday(&tv_e[i], NULL);
    }
}
```

WRITEタスク

```
void write_task () {
    ….

    /* 仮想デバイスファイルを開く */
    fd = open(“/dev/irqhook”, O_RDWR);

    ts.tv_sec = 0;
    ts.tv_nsec = 16 * 1000 * 1000; /* 16ms */

    for(i=0; i<10000; i++) {
        /* 時刻を記録 */
        gettimeofday(&tv_s[i], NULL);
        /* write()を発行 */
        write(fd, &ch, sizeof(char));
        /* 16msの間スリープ */
        nanosleep(&ts, NULL);
    }
}
```

仮想デバイスドライバ

```
ssize_t irqhook_write( …. ) {
    ….
    do_gettimeofday(&tm);

    atomic_inc(&count);
    /* 待ちキューに登録されて
         いるタスクを起床 */
    wake_up(&q);
    ….
}
```

```
ssize_t irqhook_read( …. ) {
    ….
    pending = atomic_read(&count);
    while (pending == 0) {
        prepare_to_wait(&q, &wait,
                TASK_INTERRUPTIBLE);
        pending =
              atomic_read(&count);
        /* write()されるまでスリープ */
        if (pending == 0)
                    schedule();
        finish_wait(&q, &wait);
        ….
    }
    copy_to_user(bufp, &tm,
            sizeof(struct timeval));
    ….
}
```

負荷タスク

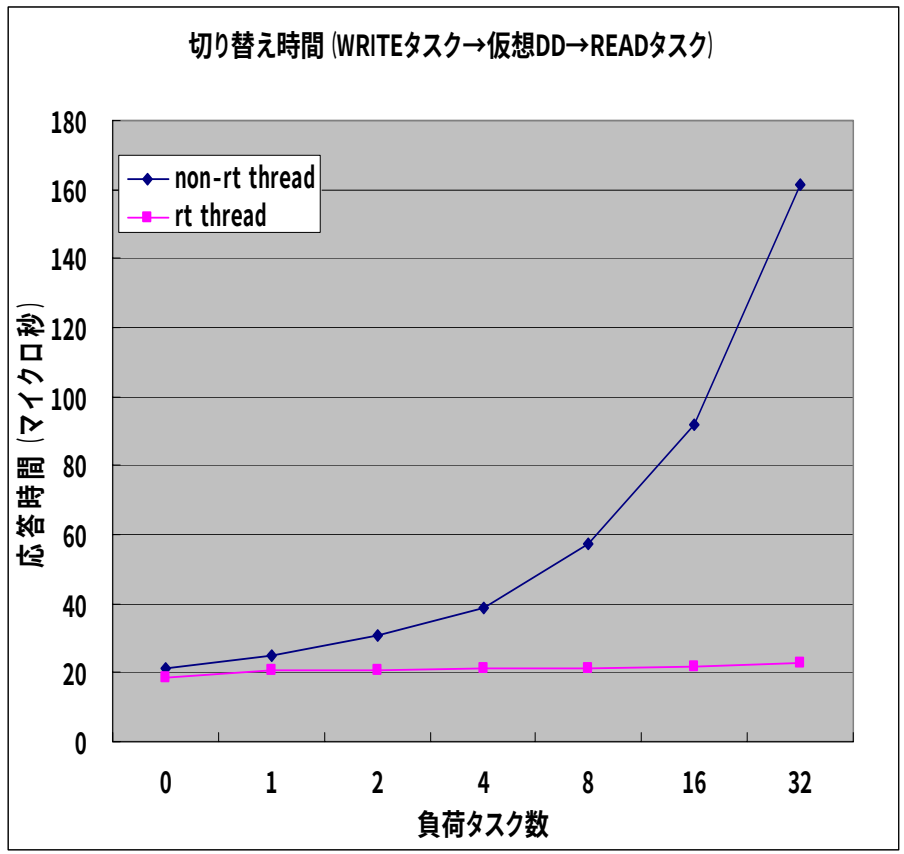
```
void busyloop() {
    ….
    ts.tv_sec = 0;
    ts.tv_nsec = 1 * 1000; /* 1us */

    while (1) {
        nanosleep(&ts, NULL); /* 1us毎に起床 */
    }
}
```

実験条件

- READタスク、WRITEタスクはそれぞれ、仮想デバイスファイルに対してread(), write()を1万回繰り返す。
- WRITEタスクは、16ms毎にwrite()を発行する。
- READタスクは、read()から戻った後、ただちに次のread()を発行する。
- WRITE()タスクでのwrite()発行時からREADタスクがread()から処理がもどってくるまでの時間を測定する。また、仮想デバイスドライバ内でwrite()処理を受け付けてからREADタスクが動き出すまでの時間も測定する。
- 測定中、1usのnanosleep()を繰り返す負荷タスクを複数個並列に走らせる。個数は、0、1、2、4、8、16、32と変化させる。
- READタスクとWRITEタスクは、通常のスレッド(non-RTスレッド) による実行とRTスレッド(優先度1、ラウンドロビンスケジューリング (SCHED_RR)) による実行の2パターン行う。
- 負荷タスクは常にNon-RTスレッドで実行する。

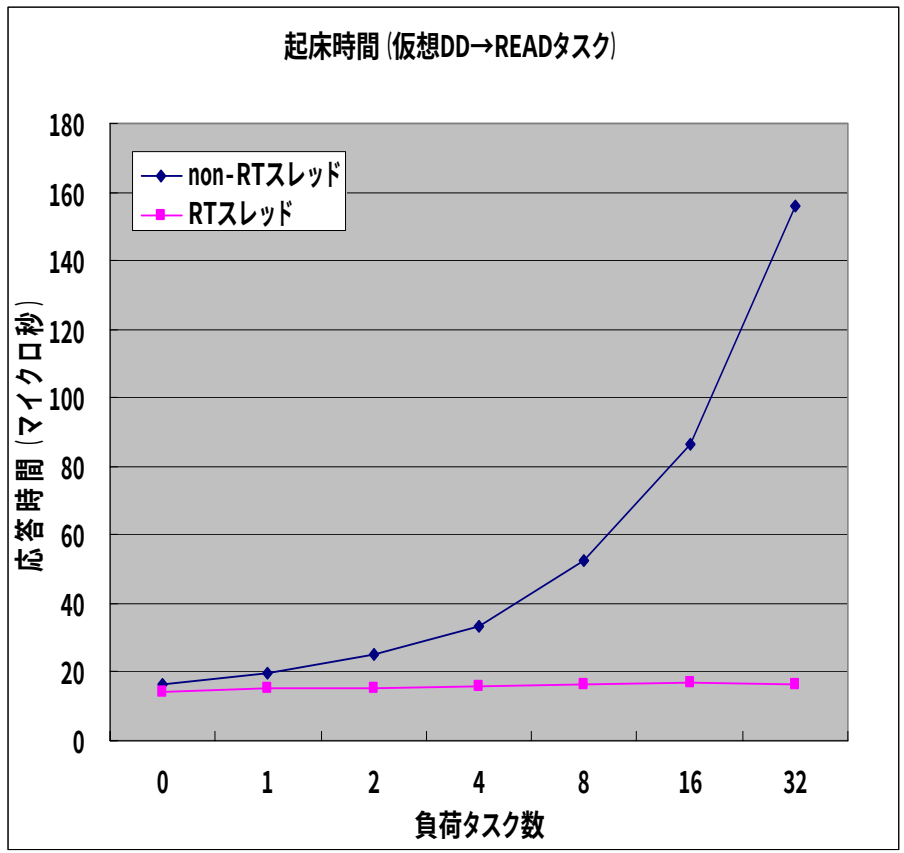
実験結果（スレッド切り替え時間）

- RTスレッドによるタスク実行は、負荷タスクの個数にかかわらずほぼ一定
- Non-RTスレッドの場合は、負荷タスクの数に比例



実験結果（スレッド起床時間）

- RTスレッドの場合はほぼ一定
- Non-RTスレッドのスレッド切り替えオーバヘッドは、起床時間と比例



考察

- RTスレッドでは、安定した応答時間が得られた。これは、RTスレッドが、負荷タスク等のNon-RTスレッドの状況にかかわらず、ただちにディスパッチ対象となるためである。
- Non-RTスレッドは他タスクの影響を大きく受ける。

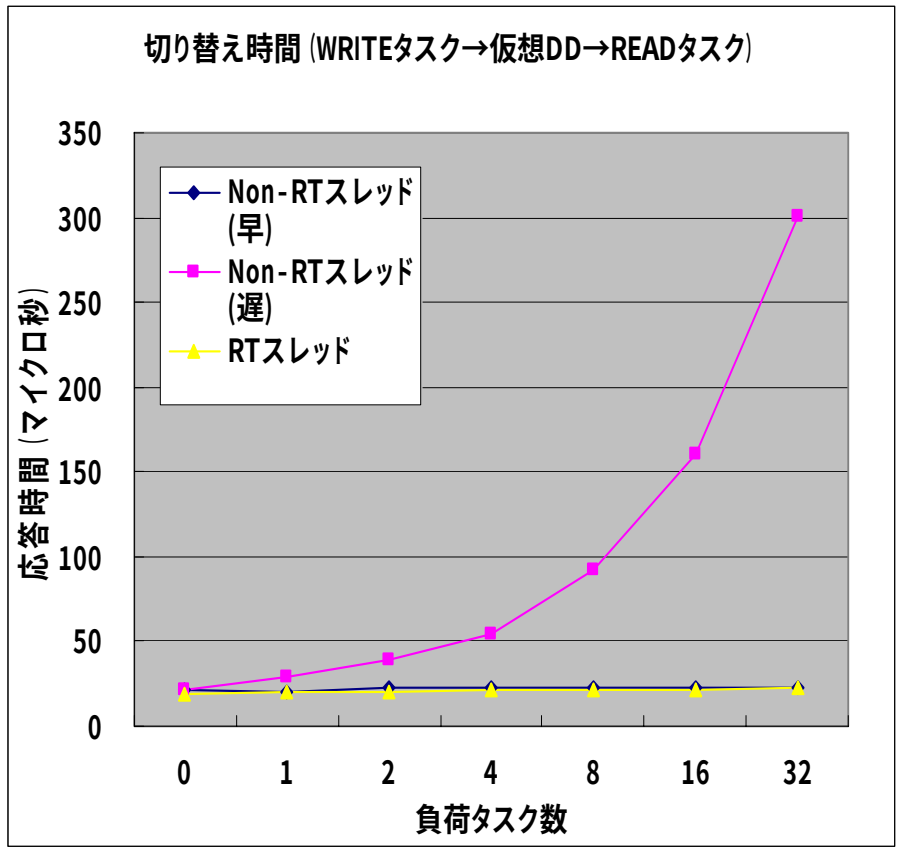
RTスレッドはユーザレベル・デバドラの実現に有効！
Non-RTスレッドは使えない？

Non-RTスレッドの挙動

- Non-RTスレッドの測定結果に変わった挙動が‥

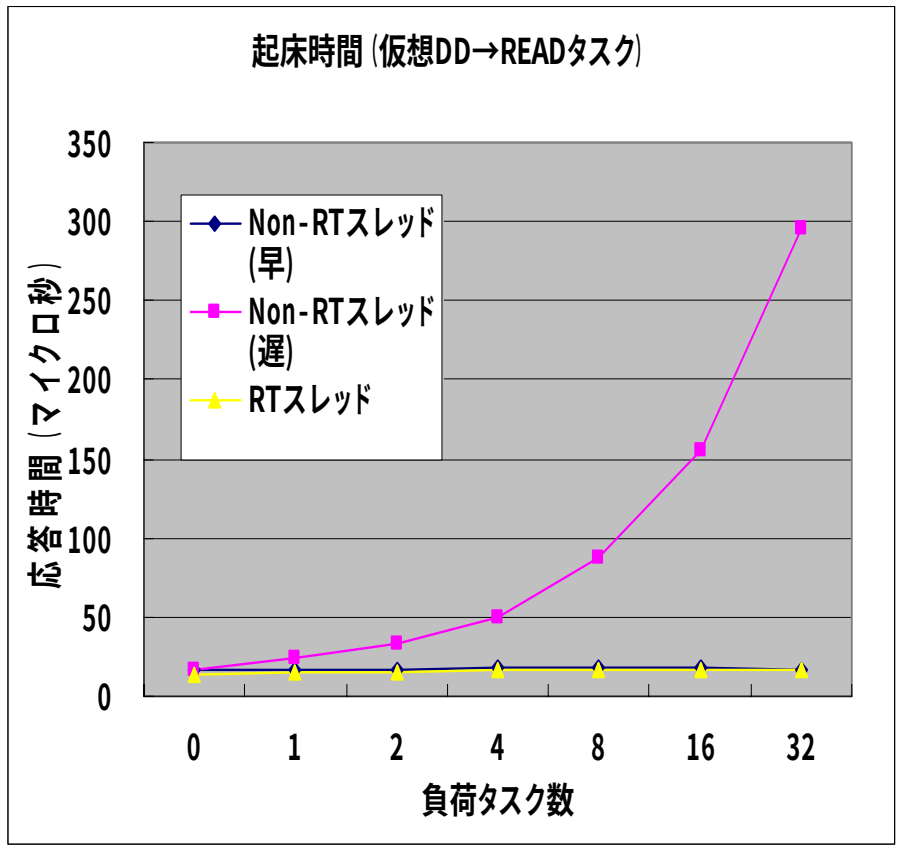
実験結果（Non-RTスレッドの切り替え時間を早遅で分離）

- Non-RTスレッドの切り替え時間が早いとき、RTスレッドとほぼ同じ応答時間
- 遅い場合は、負荷タスクの数に比例



実験結果（Non-RTスレッドの起床時間を早遅で分離）

- 起床時間も同様の結果



Non-RTスレッド(負荷タスクx 16)の測定結果

ループ回数	切り替え時間	応答時間
0	94	185
1	17	23
2	18	23
3	17	21
4	17	22
5	16	22
6	17	22
7	155	160
8	17	22
9	154	159
10	17	22
11	154	160
..	..	..

RTスレッド(負荷タスクx 16)の測定結果

ループ回数	切り替え時間	応答時間
0	91	163
1	17	23
2	16	21
3	16	21
4	17	21
5	16	21
6	16	20
7	16	21
8	16	20
9	16	21
10	16	20
11	16	21
..	..	..

疑問

- Non-RTスレッドの応答時間は、最初数回だけ安定しているのに、その後、早い遅いの交互に変化するのはなぜ？
- RTの有無に関わらず、1回目だけ極端に遅いのはなぜ？

ディスカッション

- みなさんはどう思われますか？ご意見を聞かせてください。
  - それはXXXが原因に違いない。
  - こんな実験をすればいいんじゃないか。
  - 実験にはこんなツールが使えるよ。
  - 実験（結果）が間違っている。
  - などなど‥

個人的推測

- Non-RTスレッドの起床時間が交互に変わる理由
  1. なんらかの原因で、負荷タスクとREADタスクの優先度が同じになる。
  2. 一度起床すると、次にランキューに入るときに、負荷タスクの後ろに入るため、遅れる。
  3. 一度遅れるとスリープ時間が長くなり、次の起床時、優先的にディスパッチされる。

  ・・・・かな？？
- 最初数回だけ安定する理由
  - ？
- 1回目だけ極端に遅い理由
  - ？？