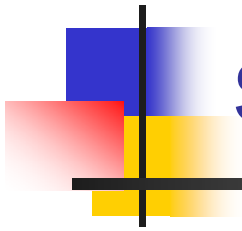
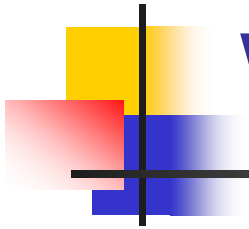


Embedded optimization (2) Starvation free real time scheduler

NEC
システムプラットフォーム研究所
塚本　明 (Akira Tsukamoto)

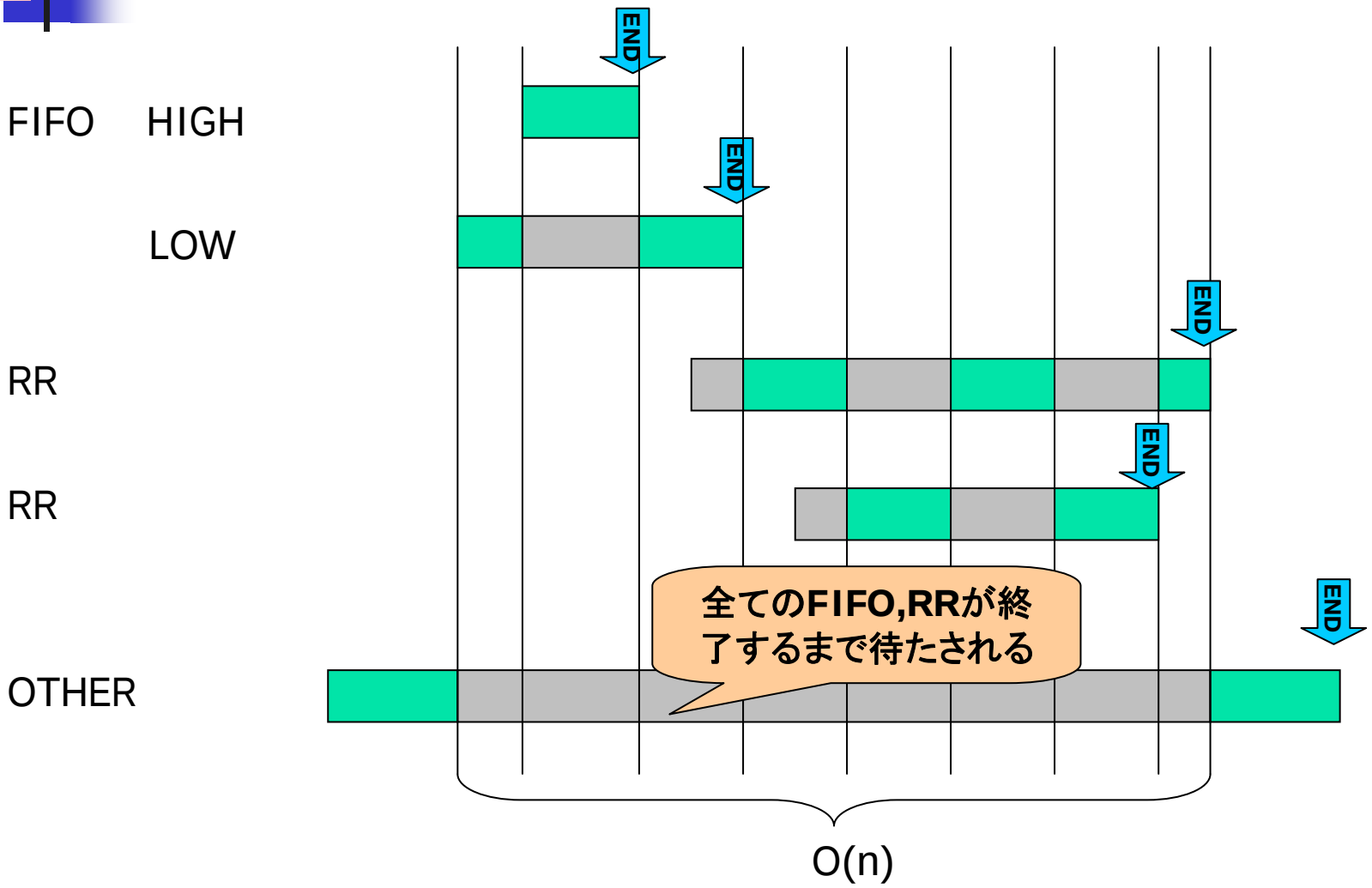


WRR スケジュラーの設計

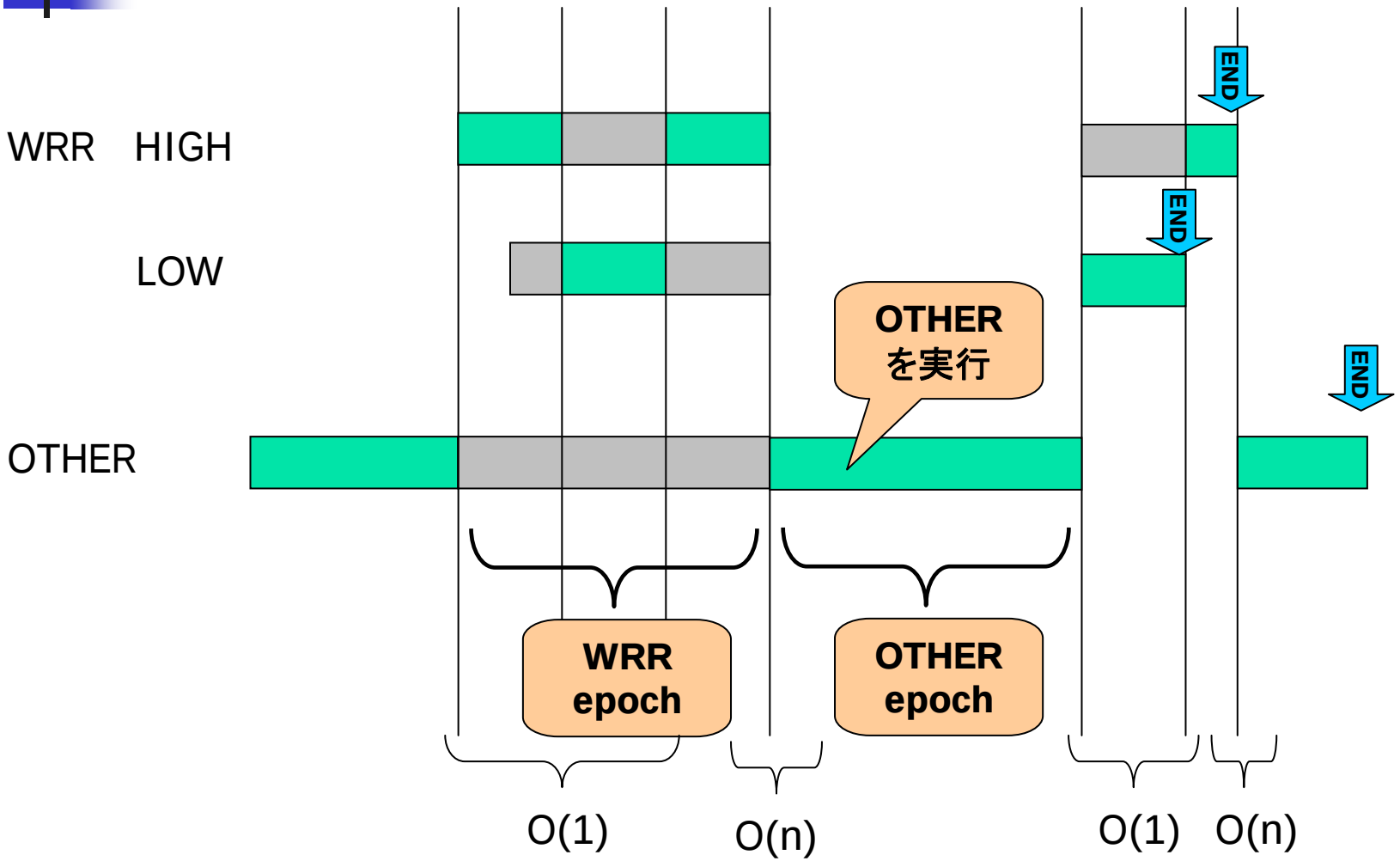
- No starvation
  - 従来の Linux スケジュラーではリアルタイムタスク(FIFO,RR)が存在中は、全く通常タスク(OTHER)が選択されない。(タスク切換えが起きずOTHERタスクはずっと待たされる)
  - 新規に導入した WRR スケジュラーでは WRR タスクが存在する時でも、OTHER タスクが実行される。
- Weighted Round Robin
  - プライオリティーに従ってタスク毎の実行時間の調整が可能なラウンドロビンアルゴリズム
- Light weight
  - Kernel 2.4 までのスケジュラーは O(n) アルゴリズム
  - 本方式では、WRR タスク存在時に
    - OTHER と WRR のタスクキュー間の切り替えを O(1) で実現
    - RR、FIFO タスクの挿入時にO(1) で WRR タスクキューから切り替え
  - 追加コード 200 行程
    - Kernel 2.4 までの Linux オリジナルアルゴリズムのスケジュラーがベース

方式ついては、後述のパッチを参考に説明

Task selecting, conventional (kernel 2.4)



Task selecting, with WRR



Setting priority Example

- プロセス起動

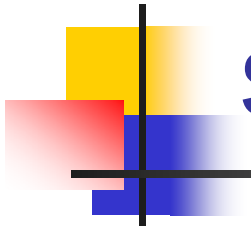
```
for(x=0;x<NUM_PROGS;x++) {
    /* fork a new process and save the pid */
  pids[x] = fork();

  /* if you are a forked program, go into infinite */
  if(!pids[x]) {
      infiniteLoop();
  }
}
```

- プライオリティーの設定

```
for(x=0; x<5; x++) {
      sParam.sched_priority = 1 + x;
      sched_setscheduler(pids[x], SCHED_WRR, &sParam);
}
```

数字が大きくなるほど該当タスクの実行時間が長くなる

Implementation sched.h sched.c



- Modified functions
  - goodness()
    - runqueue の中で優先度の高いタスク(goodness値)を計算する時に呼ばれる関数。WRR タスクの場合はgoodness値が必要無い為、計算しない処理に修正
  - try_to_wake_up()
    - sleep 状態のタスクが、runqueue に挿入される時に呼ばれる関数。FIFO, RR, WRR タスクが存在するかの判定をO(1)で行なう為に、カウンターを追記
  - schedule()
    - 実際のタスク切替え時に呼ばれる関数。詳細は下記のスライドで。
  - setscheduler()
    - タスクの属性(FIFO,RR,WRR)設定や、プライオリティー設定を行なう関数。通常タスク(OTHER)をWRR タスクに変更時に、wrr を wrr_runqueue に挿入。
- Added functions
  - 新設のwrr_runqueue 操作関数。
  - add_to_wrr_runqueue()
  - del_from_wrr_runqueue()
  - move_first_wrr_runqueue()
  - move_last_wrr_runqueue()

patch 抜粋

- 200 lines 前後
- Relatively small -> Rubust

sched.h

```
diff -urNX dontdiff linux-2.4.20-mv/include/linux/sched.h linux-2.4.20-mv-
sched/include/linux/sched.h
--- linux-2.4.20-mv/include/linux/sched.h       2002-11-29 08:53:15.000000000 +0900
+++ linux-2.4.20-mv-sched/include/linux/sched.h         2005-10-31 23:46:40.000000000 +0900
@@ -118,7 +118,8 @@
 #define SCHED_OTHER                             0
 #define SCHED_FIFO                      1
 #define SCHED_RR                        2
-
+#define SCHED_WRR               4 /* WRR added */
+
 /*
  * This is an additional bit set when we want to
  * yield the CPU for one re-schedule..
@@ -896,6 +897,16 @@
                 p->run_list.next = NULL;
 }
```

sched.c

```
+/* WRR start */
+static LIST_HEAD(wrr_runqueue_head);
+static int rt_tasks = 0;
+static int wrr_tasks = 0;
+/* WRR end */
```

sched.c try_to_wake_up()

```
@@ -365,7 +410,18 @@
                p->state = TASK_RUNNING;
                if (task_on_runqueue(p))
                        goto out;
-               add_to_runqueue(p);
+
+//             add_to_runqueue(p); /* WRR deleted */
+               /* WRR start */
+               if (p->policy == SCHED_WRR) {
+                       add_to_wrr_runqueue(p);
+                       wrr_tasks++;
+               } else {
+                       add_to_runqueue(p);
+                       if (p->policy != SCHED_OTHER)
+                               rt_tasks++;
+               } /* WRR end */
+
```

sched.c schedule()

```
@@ -591,6 +649,13 @@
                                        move_last_runqueue(prev);
                                }

+               /* WRR start */
+               if (prev->policy == SCHED_WRR)
+                       if (!prev->counter) {
+                               move_last_wrr_runqueue(prev);
+                       }
+               /* WRR end */
+
                switch (prev->state) {
                        case TASK_INTERRUPTIBLE:
                                if (signal_pending(prev)) {
```

sched.c schedule()

```
@@ -598,7 +663,17 @@
                                        break;
                                }
                        default:
-                               del_from_runqueue(prev);
+                               /* WRR start */
+                               if (prev->policy == SCHED_WRR) {
+                                       del_from_wrr_runqueue(prev);
+                                       wrr_tasks--;
+                               } else {
+                                       del_from_runqueue(prev);
+                                       if (prev->policy != SCHED_OTHER)
+                                               rt_tasks--;
+                               } /* WRR end */
+
                        case TASK_RUNNING:;
                }
                prev->need_resched = 0;
```

sched.c schedule()

```
@@ -613,13 +688,40 @@
                 */
                next = idle_task(this_cpu);
                c = -1000;
-               list_for_each(tmp, &runqueue_head) {
+               /* WRR tasks */  /* pick the first WRR task in wrr_runqueue */
+               if (rt_tasks == 0 && wrr_tasks != 0) {
+                       list_for_each(tmp, &wrr_runqueue_head) {
+                               p = list_entry(tmp, struct task_struct, run_list);
+                               if (can_schedule(p, this_cpu)) {
+                                       if (p->counter == 0) {
+                                               c = 0;  /* finished one epoch for WRR */
+                                               break;
+                                       }
+                                       next = p;       /* first WRR task */
+                                       c = WRR_SELECTED;
+                               }
+                       }
+               }
```

sched.c schedule()

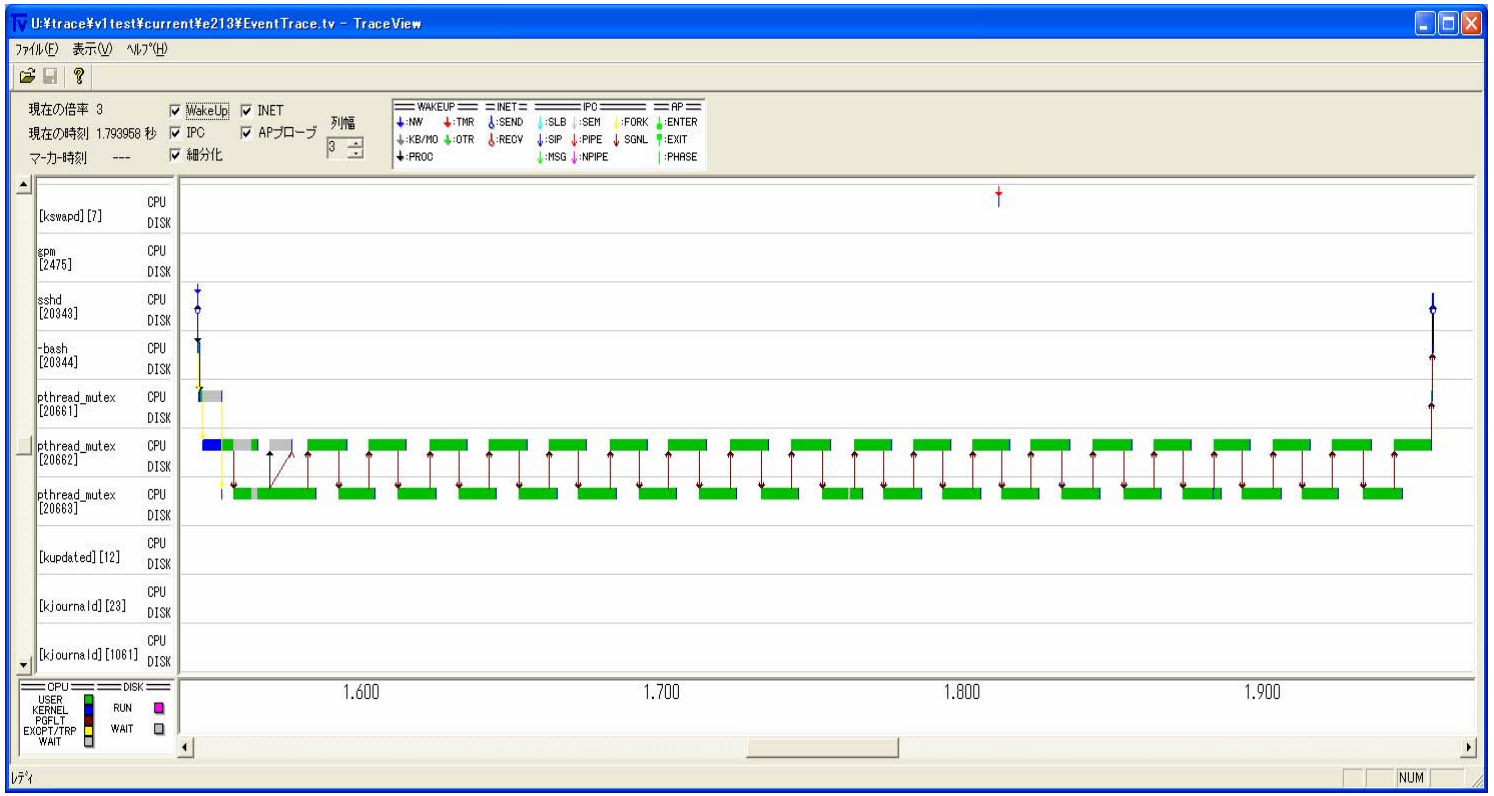
```
@@ -628,13 +728,42 @@

                        spin_unlock_irq(&runqueue_lock);
                        read_lock(&tasklist_lock);
-                       for_each_task(p)
-                               p->counter = (p->counter >> 1) + NICE_TO_TICKS(p->nice);
+                       for_each_task(p){
+                               /* WRR start */
+                               if (p->policy == SCHED_WRR)
+                                       p->counter = p->counter + p->rt_priority * 2;
+                               else
+                                       p->counter =
+                                        (p->counter >> 1) + NICE_TO_TICKS(p->nice);
+                               /* WRR end */
+                       }
                        read_unlock(&tasklist_lock);
                        spin_lock_irq(&runqueue_lock);
                        goto repeat_schedule;
                }
```

sched.c schedule()

```
@@ -965,6 +1096,21 @@
                p->policy = policy;
                p->rt_priority = lp.sched_priority;

+               /* WRR start */
+               if (policy == SCHED_WRR) {
+                       if (task_on_runqueue(p)) {
+                               move_first_wrr_runqueue(p);
+                       }
+               }
+               /* WRR end */

                current->need_resched = 1;

 out_unlock:
```

Screen Shot

実装した WRR スケジュラーのタスク切替え動作を確認
2つの WRR タスク をSystem Director Embedded によりトレースした



http://www.sw.nec.co.jp/embedded/it_infra.html