IN4343
REAL TIME SYSTEMS
Lab Assignment - III

March 26, 2018

Chinmay Pathak (4740327)
Himanshu Shah (4739779)

In the exercises of Lab 2, we observed an error in the scheduling behavior of SchedulerNP.c. We provided you with an update of the scheduler, with a file name SchedulerNP-new.c. In this new version, the guard of the inner while-loop is strengthened compared to the erroneous version. We first check the behavior of this update, i.e. whether or not the anomaly has been resolved.

1. *What has changed to the behavior of the system and why is this correct?*
   Ans: After strengthening of guard of the inner-while loop, we can see that after 2.0 sec when BlinkGreen() and BlinkYellow() both are pending, BlinkYellow() is executed first then BlinkGreen() which was not the case previously. This is correct because in this way, if a task with higher priority is pending it is executed first and then the tasks with lower priorites.

2. *Formulate a "hypothesis" about the expected behavior of the system when these two improvements of HandleTasks() are realized. In particular, address the following issues:*

   a) the consequence for the execution time of TimerIntrpt(), e.g. in terms of

   i) the delay $\Delta_{IH,1}^{ON} = f_{IH,1} - r_{IH,1}$ with activation of tasks compared to SchedulerNP-new.c;

   Ans: Since TimerIntrpt() contains an extra if condition in the improved code, the $\Delta_{IH,1}^{ON}$ should be greater in improved code.

   ii) the delay $\Delta_{IH,2}^{ON} = f_{IH,2} - r_{IH,2}$ without activation of tasks compared to SchedulerNP-new.c;

   Ans: For the case of without activation of task, since the code remains same, the execution time of improved code and SchedulerNP-new.c should be same.

   b) the consequence for the time needed to execute `CountDelay(60000)`.

   Ans: The time needed to execute `CountDelay(60000)` should also increase as the duration of TimerIntrpt() is going to be longer.

3. Before actually implementing your solution, you are expected to answer the following two questions.

   a) Formulate an invariant, i.e. a mathematical relation that is "always" true, describing the relation between the variable Pending and HPrioPendingTask.

   Ans: If any task is pending HPrioPendingTask is equal to task priority but pending is equal to 1. i.e.

`HPrioPendingTask > -1 ⇔ Pending = 1`

If no tasks are pending then Pending is zero and HPrioPendingTask is -1 i.e.
`HPrioPending = -1 ⇔ Pending = 0`

b) Describe which changes are required where in the code.

Ans: The following changes must be made in the code:

- Since both HandleTasks() and TimerIntrpt() are using HPrioPendingTasks variable it must be made mutually exclusive. So, interrupt is disabled when HandleTask() accesses HPPrioPendingTasks variable and is enabled again later.
- The while loop in the HandleTask() should be changed to run only the number of times of the priority of the task and not the size of task array.
- The condition of the inner while loop of HandleTask() should be changed such that it is terminated only when a higher priority task is pending to be executed. This is done by changing the condition as `while(i>=0 && i>HighPrioPending)`

Subsequently implemented improvements, and answer the following two questions.

c) Include a "diff -u" between SchedulerNP-new.c and your new implementation in the report.

Ans:

```
--- SchedulerNP-new-original.c  2017-04-20 16:28:04.000000000 +0200
+++ SchedulerNP-new-improved.c  2018-03-26 18:07:40.580315350 +0200
@@ -48,8 +48,7 @@
 #include "Scheduler.h"

 Task Tasks[NUMTASKS];           /* Lower indices: lower priorities
*/
-uint8_t Pending = 0;            /* Indicates if there is a pending task
*/
-
+int8_t HPrioPendingTasks = -1;          /* Indicates if there is a pending
*/
 uint16_t IntDisable (void)
```

```
 {
   uint16_t sw;
@@ -120,11 +119,14 @@

 void HandleTasks (void)
 {
-  while (Pending) {
-    int8_t i=NUMTASKS-1; Pending = 0;
-    while (i>=0 && !Pending) {
+  while (HPrioPendingTasks >= 0) {
+       uint16_t sw = IntDisable();
+       int8_t i=HPrioPendingTasks; HPrioPendingTasks = -1;
+       RestoreSW(sw);
+       _EINT();
+       while (i>=0 && i>HPrioPendingTasks) {
       Taskp t = &Tasks[i];
-      if (t->Activated != t->Invoked) {
+         if (t->Activated != t->Invoked) {
         if (t->Flags & TRIGGERED) {
           t->Taskf(); t->Invoked++;
         }
@@ -139,14 +141,16 @@
   do {
     Taskp t = &Tasks[i];
     if (t->Flags & TRIGGERED) { // countdown
-      if (t->Remaining-- == 0) {
-        t->Remaining = t->Period-1;
-        t->Activated++;
-       Pending = 1;
-      }
+       if (t->Remaining-- == 0) {
+          t->Remaining = t->Period-1;
+          t->Activated++;
+                  if(i>HPrioPendingTasks){
+                        HPrioPendingTasks = i;
```

```
+                       }
+                   }
        }
    } while (i--);
-   if (Pending) ExitLowPowerMode3();
+   if (HPrioPendingTasks>=0) ExitLowPowerMode3();
 }

 #endif
```

d) Discuss the correctness of your solution .

Ans: The proposed improvement in the code successfully handles the race condition between Handletask() and TimerIntrpt() by disabling the interrupt when HandleTask() accesses the 'HPrioPendingTask' variable and once it has updated the variable interrupt is enabled again.

To validate the hypothesis, the consequences for the executions of TimerIntrpt() and CountDelay(60000) are to be measured.

4. Measure the consequences of the improvements of HandleTasks() for

   a) the execution time of TimerIntrpt(), i.e. in terms of

   i) the delay $\Delta^{ON}_{IH,1} = f_{IH,1} - r_{IH,1}$ with activation of tasks compared to SchedulerNP-new.c;

   Ans: The delay $\Delta^{ON}_{IH,1}$ with activation of tasks was observed to be 515.77 μs for SchedulerNP-new.c and 558.465 μs for improved code.

   ii) the delay $\Delta^{ON}_{IH,2} = f_{IH,2} - r_{IH,2}$ without activation of tasks compared to SchedulerNP-new.c;

   Ans: The delay $\Delta^{ON}_{IH,2}$ without activation of tasks was observed to be 462 μs for schedulerNP-new.c and 497 μs for improved code.

   First, the behavior of SchedulerNP-new.c and the improved version are to be visualized. The improvements are subsequently measured.

5. Visualize the behavior of SchedulerNP-new.c and the improved version for

a) the start-up situation;

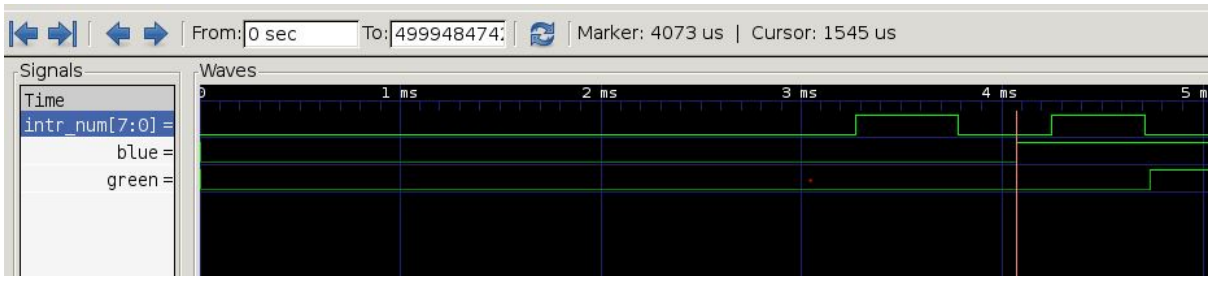

Figure 1: schedulerNP-new.c startup scenario

In case of improved code the execution of task happens earlier as compared to

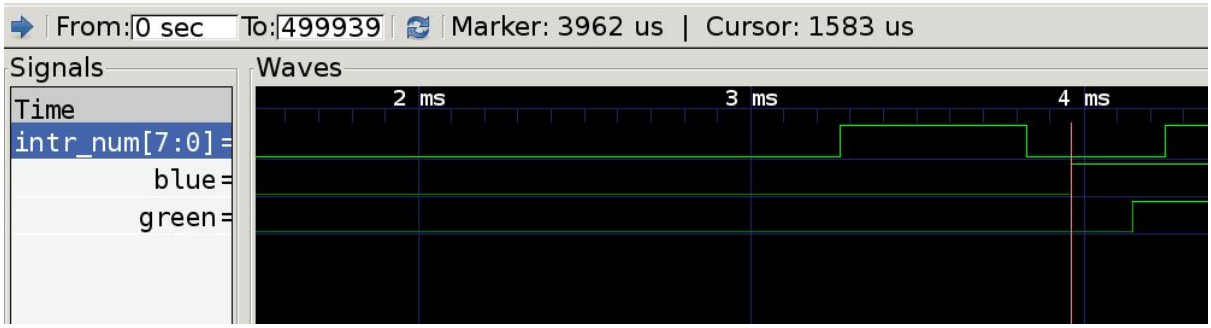

Figure 2: schedulerNP-new improved startup scenario

SchedulerNP-new.c. We can see for SchedulerNP-new.c, tasks are executed at 4073 μs whereas for improved code same task is executed at 3962 μs, a reduction of 111 μs.

b) the situation immediately after CountDelay(60000).

We noticed two things here, `CountDelay(60000)` executes for longer time(107ms

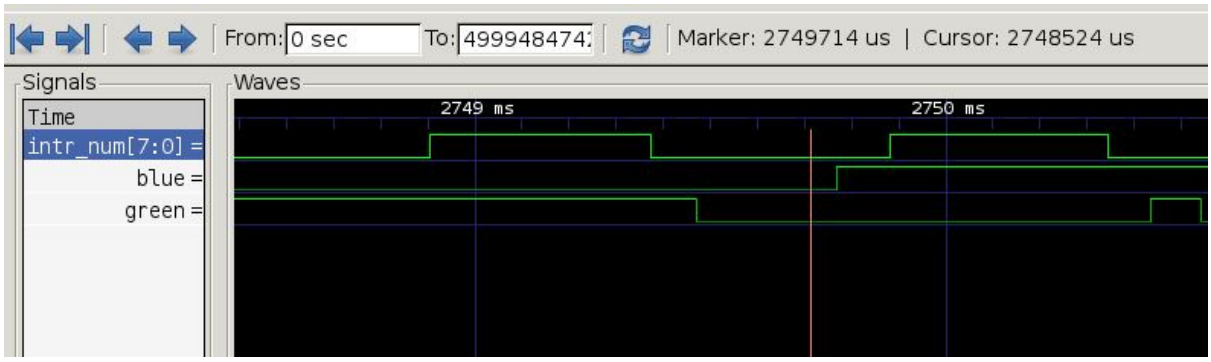

Figure 3: schedulerNP-new.c situation immediately after count delay

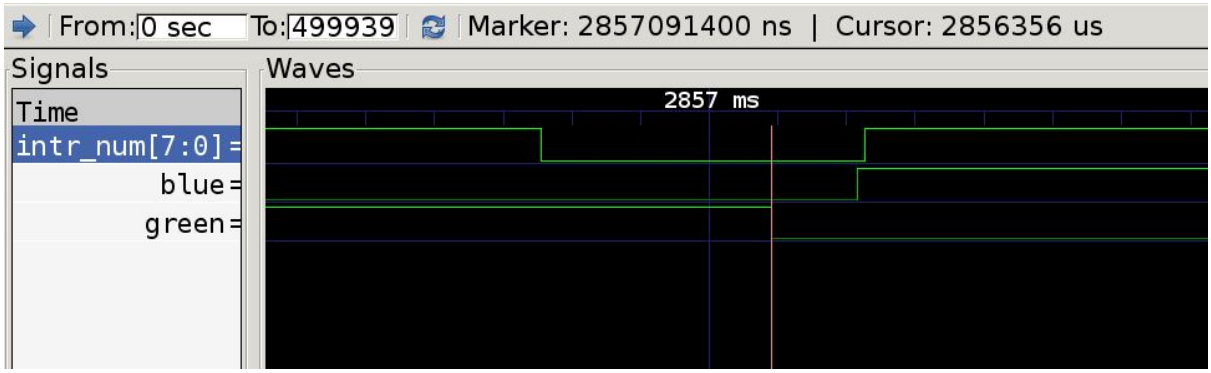

Figure 4: schedulerNP-new improved situation immediately after count delay

longer) and the handletime for execution of BlinkYellow() is reduced(173 μs earlier).

6. Measure the improvement of the efficiency of Handletasks() of your implementation compared to SchedulerNP-new.c by determining

   a) the reduction of the event latency for BlinkGreen() and BlinkYellow(), addressing item (i) at the start of Section 2 above;

   Ans: The event latency for BlinkGreen() was 404μs for SchedulerNP-new.c whereas it was 156μs. Hence, a reduction of 248μs.The event latency for BlinkYellow() was 292μs for SchedulerNP-new.c whereas it was 107μs. Hence, a reduction of 185μs.

   b) the reduction of the execution time to handle BlinkYellow() after the fourth activation of BlinkGreen(), i.e. after non-pre-emptive execution of CountDelay(60000), addressing item (ii) above.

   Ans: The time to handle BlinkYellow() for SchedulerNP-New.c was 298μs whereas for the improved code it was 125μs. Hence, a reduction of 173μs.

7. Given the measurement results in the previous question, reflect on the hypothesis and the experiment.

   Ans:

   - According to the hypothesis and as the measurements were observed the event latency for the tasks reduced because in the improved version in the handle tasks routine the highest priority activated task is directly invoked, instead of iterating through the whole tasks array from NUMTASKS-1.
   - In the second measurement of the previous question the execution time for Blinkyellow() after the delay is reduced because the inner while loop in handle task is terminated as soon as the higher priority task is pending.
   - The duration of the timer handler during no task activation should have remained same according to the hypothesis, but an extra push operation was observed in the assembly which caused the difference in improved version.