ECE 443 - Project #4

Collin Heist

October 15, 2019

Contents

1 Tracealyzer Utilization **1**
1.1 Heartbeat Tracealyzer 1
1.2 Change Notice Tracealyzer 1
1.3 LCD Tracealyzer 3

2 Testing & Verification **3**

3 Figures **4**

Listings

1 Heartbeat Task 1
2 Change Notice ISR 1
3 Change Notice Handler Task 2
4 LCD Task 3

1 Tracealyzer Utilization

1.1 Heartbeat Tracealyzer

To begin with, the project guidelines give pretty clear instructions on what components to add to the project. We're told (quite explicitly) to add a change-notice interrupt generation task, a change notice handler task, an LCD write task, and a LEDA toggling task. I chose to implement my IR sensor reading as a separate *library* file.

Because of the clear segmentation of the project, I decided to create three Tracealyzer channels. I have one for logging change notice events, another for logging LCD related events, and a final channel for logging the heartbeat task events.

The (presumably) easiest to implement was the heartbeat task. This is a zero-priority task that utilizes a non-API call way of delaying itself (so that it's always *ready*). This is my code for that:

Listing 1: Heartbeat Task

```
static void task_leda_toggle(void* task_params) {
  unsigned int t_wait, t_start;

  for (;;) {
    t_start = ReadCoreTimer();
    t_wait = CORE_MS_TICK_RATE * LEDA_TOGGLE_MS;
    while (ReadCoreTimer() - t_start < t_wait);
    LATBINV = LEDA;
    if (configUSE_TRACE_FACILITY)
      vTracePrint(trace_leda_toggle, "Toggled LEDA");
  }
}
```

I utilized Tracealyzer here to verify that the toggling task was executing properly. However, this was a bit superfluous as Tracealyzer automatically logs any task-switches, resulting in a log of the toggle anyway. However, after this allowed me to verify that my heartbask task was executing properly (on its own).

1.2 Change Notice Tracealyzer

Next, I added in the change notice functionality. We were told to add a change notice ISR that was triggered by a task that sets the change notice flag every 5 milliseconds. I utilized my Tracealyzer channel, **trace_cn**, to note the important events in this sequence of events. For starters, inside the handler is the code shown in **Listing 2**.

Listing 2: Change Notice ISR

```
void isr_change_notice_handler() {
  portBASE_TYPE move_to_higher_priority = pdFALSE;

  if (configUSE_TRACE_FACILITY)
    vTracePrint(trace_cn, "Giving semaphore from CN ISR");
```

```
    xSemaphoreGiveFromISR(cn_semaphore, &move_to_higher_priority);

    mCNClearIntFlag();
    mCNOpen(CN_OFF, (CN8_ENABLE), 0);

    portEND_SWITCHING_ISR(move_to_higher_priority);
  }
```

As you can see, I used Tracealyzer here to notify when the semaphore is being given, which (ideally) should prompt a transition into the change notice handler task (shown in **Listing ??**).

Listing 3: Change Notice Handler Task

```
static void task_change_notice_handler(void* task_params) {
  float temp = 0.0;
  char lcd_message[LCD_WIDTH+1] = {0};
  portBASE_TYPE queue_status;

  xSemaphoreTake(cn_semaphore, 0);
  for (;;) {
    xSemaphoreTake(cn_semaphore, portMAX_DELAY);
    if (configUSE_TRACE_FACILITY)
      vTracePrint(trace_cn,
      "Received semaphore, reading from IR sensor");

    temp = read_ir_temp();
    if (temp == ERROR_TEMP) {
      if (configUSE_TRACE_FACILITY)
        vTracePrint(trace_cn, "Error in temperature reading");
    }
    else {
      if (configUSE_TRACE_FACILITY)
        vTracePrint(trace_cn,
          "Temperature read successfully - adding to Queue");
      sprintf(lcd_message, "Temp: %3.1f", temp);

      queue_status = xQueueSendToBack(lcd_string_queue,
        &lcd_message, portMAX_DELAY);
      if (configUSE_TRACE_FACILITY && queue_status == errQUEUE_FULL)
        vTracePrint(trace_cn,
          "Unable to add to Queue, Queue is full");
    }
  }
}
```

Once again, I used the change notice Tracealyzer channel to mark significant events as they occur. In this instance that is receiving the semaphore, reading from the IR sensor,

whether there was an error in the temperature reading or not, and if adding to the queue went properly. This allowed me to easily see the sequence of events occuring when a read was initiated by the change notice interrupt being set. This sequence of events is particularly evident in my Tracealyzer output, where the various channels can be seen handing off to each other. See this sequence of events inside **Figure 1**.

1.3 LCD Tracealyzer

Finally, the LCD operations had their own Tracealyzer channel. This was used entirely by the task being used to write to the LCD, and only notified the user when a pointer was received from the Queue, and when the message was written to the LCD. Again, this is shown below.

Listing 4: LCD Task

```
static void task_display_lcd(void* task_params) {
  char lcd_message[LCD_WIDTH+1] = {0};
  unsigned int i = 0;
  for (;;) {
    xQueueReceive(lcd_string_queue, &lcd_message, portMAX_DELAY);
    if (configUSE_TRACE_FACILITY)
      vTracePrint(trace_lcd,
        "Received pointer from Queue - writing to LCD");

    set_cursor_LCD(FIRST_LINE_START);
    put_string_LCD(lcd_message);
    if (configUSE_TRACE_FACILITY)
      vTracePrint(trace_lcd,
        "Message written to LCD, waiting for value in Queue");

    for (i = 0; i < LCD_WIDTH + 1; i++)
      lcd_message[i] = '\0';
  }
}
```

2 Testing & Verification

Adding these Tracealyzer channels made verifying my program's behavior very easy. In particular, it showed me a problem with the heartbeat task. The Tracealyzer view showed that the LEDA toggling task was only being executed every 5 milliseconds – when the project requirements asked for 3 milliseconds. Luckily, the Tracealyzer event log also revealed why this was happening, as well. It was evident that the low-priority heartbeat task was being executed enough, but was being interrupted by the higher-priority change-notice generator, and LCD writing tasks. These tasks take a significant amount of time (on the order of 2.5 milliseconds, according to Tracealyzer), and therefore prolonged the heartbeat task from toggling by that length of time.

Given the design restrictions, there is no fix to this. The clock frequency of the IR sensor is maxed at 100 kHz, and still the writing simply takes too long. But, this did

showcase the utility of Tracealyzer, as I would not have easily been able to diagnose this problem without it.

3 Figures

In the below figure, it is quite evident that the desired sequence of events are occurring as expected. To begin, the ISR is entered for a *very* small amount of time, prompting the transfer to the change notice handler, and then the change notice task.

Inside the ISR task (it can be seen on the left side of the **Figure**, the red Actor), the semaphore is given and then a **vTaskDelayUntil()** is called. After this the change notice handler, the yellow Actor, takes the semaphore, reads the temperature and adds the string pointer to the queue. Finally, the display task (the green task), receives from that queue and the message is written to the LCD.

I've included the vertical trace view of the LCD writing operation to showcase how long it takes to perform the write. Just over 2 milliseconds are spent *just* writing to the LCD, and given the additional overhead for generating the change notice, reading from the IR sensor, and adding to the queue, it's quite clear that the desired 3 millisecond heartbeat toggle is unachievable in this lab (without adding pauses during the operation of the LCD writing task).

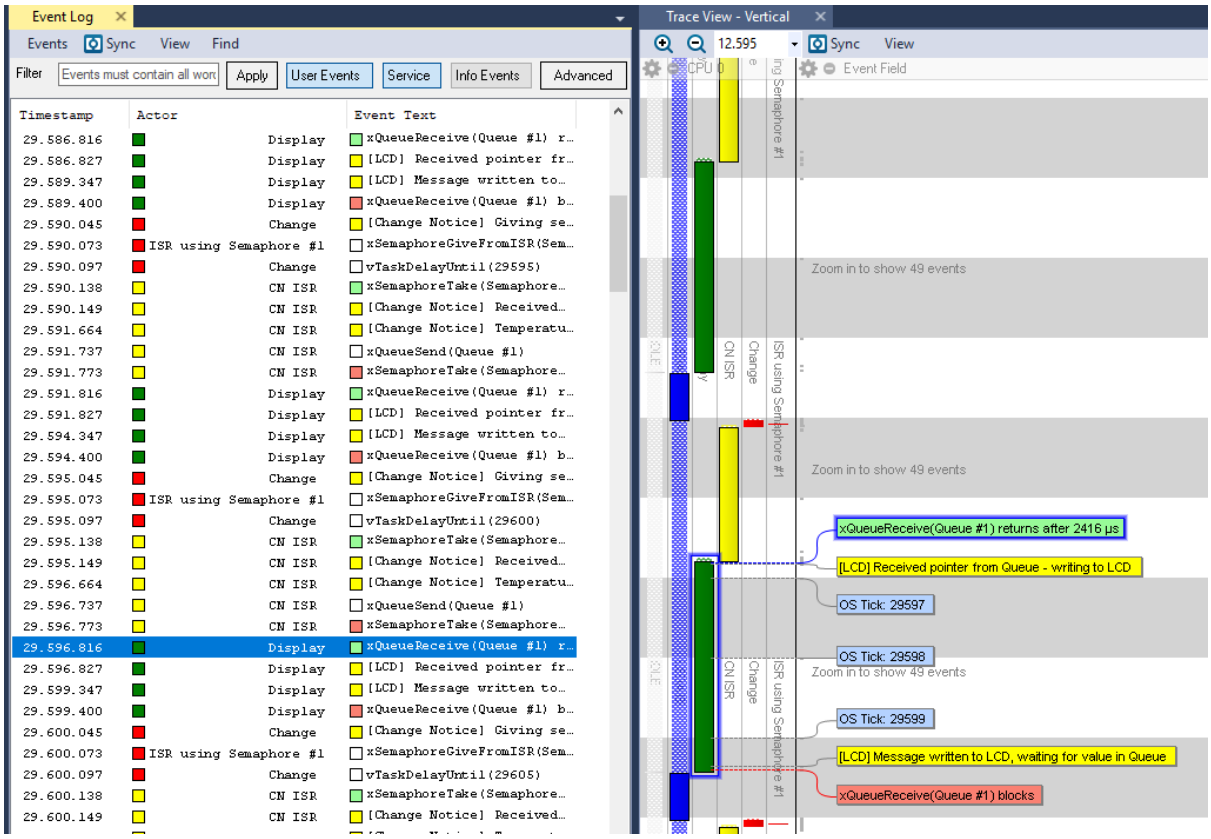

Figure 1: TraceAlyzer log of the program's operation