Lab 2

by
Garett Loghry

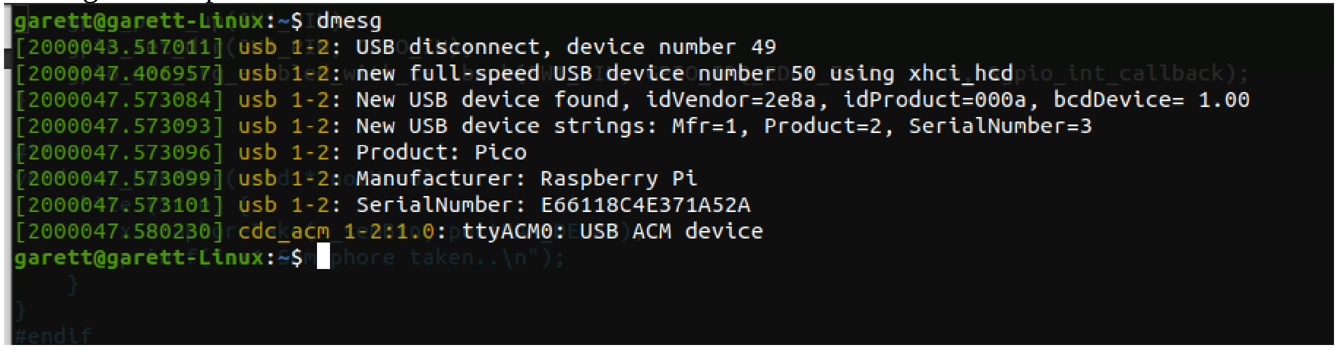
CS466 – Embedded Systems
LABORATORY REPORT

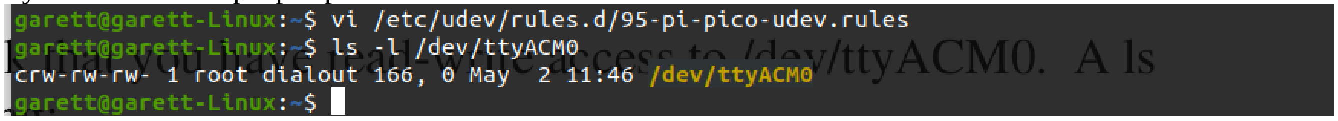
Computer Science, Washington State University Vancouver
May 2nd, 2023

Objective:
The objective of this lab was to get use familiar with using a semaphore ISR (Interrupt Service Routine) style of programming, rather than the round robin type of programming we did in lab 1. We will use FreeRTOS for their semaphores, and will continue to use that os. We also get used to debugging using a serial channel with commands such as `printf()`.

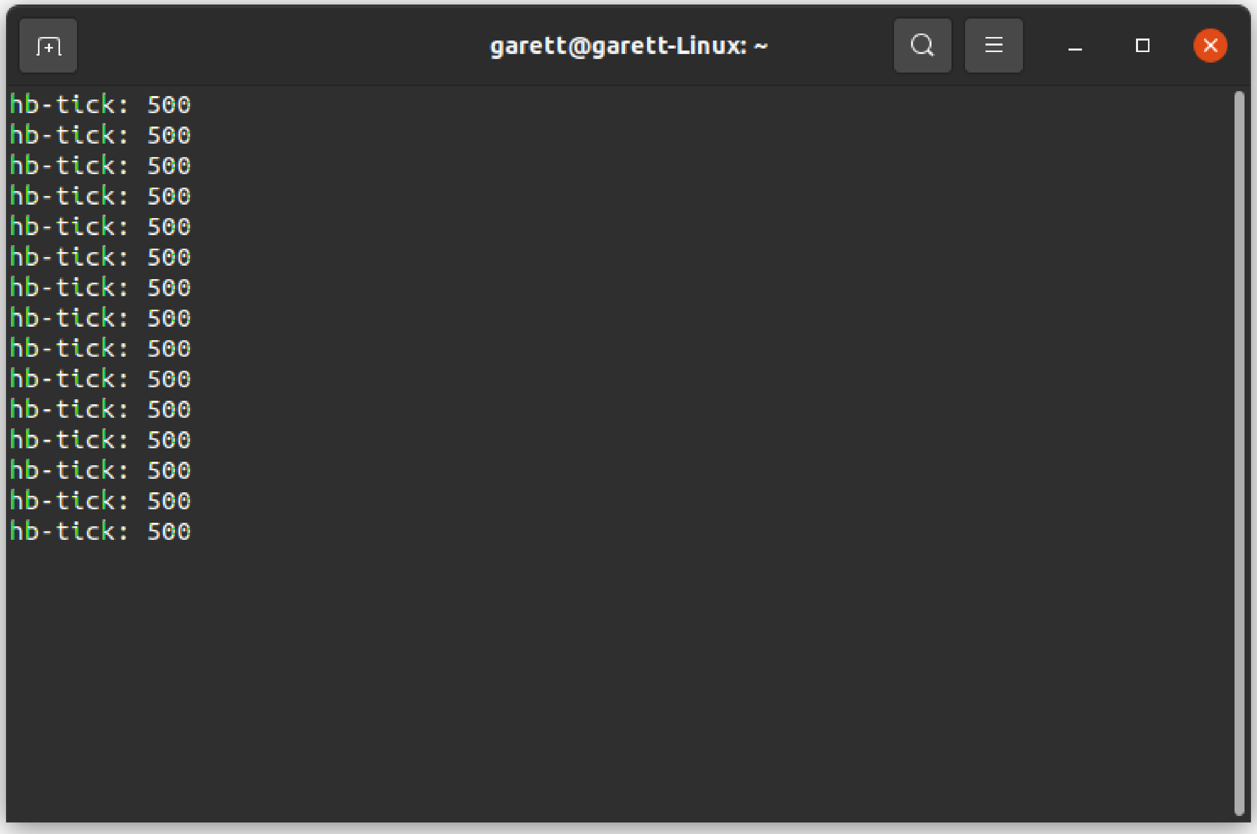
Apparatus:
- Raspberry Pi Pico
- Breadboard (protoboard)
- 3 DIP buttons
- Wires
- Laptop for power and debugging

Method:
I approached this lab in 3 major steps.
1. Enabling the serial port to communicate with the pico
    - Using `dmesg` to find the pico hardware information, adding the pi to the vendor list, and using screen to view the serial data
2. Create the semaphores and create their corresponding tasks
    - Using Millers commented code, expand the program for two button use.
3. Go through the necessary logic to achieve the required functionality
    - Blink lights according to directed frequency, add a debouncing wait, and add an alarm if no buttons being pressed after 60s.

Data:
`dmesg` shows pico and its hardware details.

```
garett@garett-Linux:~$ dmesg
[2000043.517011] usb 1-2: USB disconnect, device number 49
[2000047.406957] usb 1-2: new full-speed USB device number 50 using xhci_hcd
[2000047.573084] usb 1-2: New USB device found, idVendor=2e8a, idProduct=000a, bcdDevice= 1.00
[2000047.573093] usb 1-2: New USB device strings: Mfr=1, Product=2, SerialNumber=3
[2000047.573096] usb 1-2: Product: Pico
[2000047.573099] usb 1-2: Manufacturer: Raspberry Pi
[2000047.573101] usb 1-2: SerialNumber: E66118C4E371A52A
[2000047.580230] cdc_acm 1-2:1.0: ttyACM0: USB ACM device
garett@garett-Linux:~$
```

ttyACM0 has the proper permissions.

```
garett@garett-Linux:~$ vi /etc/udev/rules.d/95-pi-pico-udev.rules
garett@garett-Linux:~$ ls -l /dev/ttyACM0
crw-rw-rw- 1 root dialout 166, 0 May  2 11:46 /dev/ttyACM0
garett@garett-Linux:~$
```

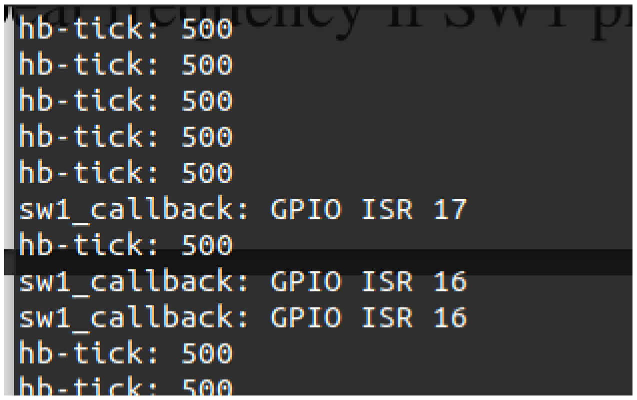
Heartbeat tick is running.



garett@garett-Linux: ~

```
hb-tick: 500
hb-tick: 500
hb-tick: 500
hb-tick: 500
hb-tick: 500
hb-tick: 500
hb-tick: 500
hb-tick: 500
hb-tick: 500
hb-tick: 500
hb-tick: 500
hb-tick: 500
hb-tick: 500
hb-tick: 500
hb-tick: 500
```

Getting interrupts from GPIO’s



```
hb-tick: 500
hb-tick: 500
hb-tick: 500
hb-tick: 500
hb-tick: 500
sw1_callback: GPIO ISR 17
hb-tick: 500
sw1_callback: GPIO ISR 16
sw1_callback: GPIO ISR 16
hb-tick: 500
hb-tick: 500
```

Results and Analysis

Lab Question 3:

a. The program functions like a single button version of lab 1. The LED flashes at 1hz with a 50% duty cycle, until switch 1 is pressed, in which case it moves to a 2hz flash with 50% duty cycle.

b. The button has no debouncing and in my opinion, is inconsistent for that reason.

Lab Question 10:

As per FreeRTOS' own website "The RTOS demo applications all use a tick rate of 1000Hz. This is used to test the RTOS kernel and is higher than would normally be required. "

Enabling the serial port with `dmesg` proved to be a little bit of a hassle because it kept picking up another program, discord, which for some reason makes hundreds of calls to dmesg, making it impossible to track down the pico correctly. Eventually, killing discord to find the pico was the best/only solution, and once `/dev/ttyACM0` could be interacted with, discord was no issue. I chose to use `screen` rather than `kermit` simply because I had used screen before for a different project, and was interested to learn more about it's functionality.

Creating the semaphores went smoothly, except I didn't realize I had to create the task before it would run. Once that was implemented correctly, I moved on quickly. Miller's commented examples helped quite a lot.

Getting through the logic was probably the toughest part, and I'm still not fully happy with it's implementation. I was having issues with lower priority interrupts coming through still when I had multiple buttons pressed, for example. This was solved by suspending the task, though this feels more like a bandaid than a fix.

The buttons will still bounce if pressed down instead of just tapped. While the semaphore will only flag the button once while pressed, the button may retrigger on this new bounce. This is assumed to be fine since the instructions set a time of 25mns to allot for debouncing, and to go further would require more experience than I likely am able to provide right now.

Finally, it's hard to know based on the directions given if the lights should flash their directed time regardless of a different button press or not. I believe my program handles multiple different button presses reasonably because they do flash their directed amount of time, even if there's some overlap between the two.

Conclusion:

I learned about semaphores, the FreeRTOS semaphore system, communication over serial ports, and got better at programming with the pico. Overall I spent about 5 hours on this lab, and since I'm behind, I need to move on, otherwise I might debug more. I am happy with the final results and would present this to an employer if asked.

```
/**
 * @brief CS466 Lab1 Blink proigram based on pico blink example
 *
 * Copyright (c) 2020 Raspberry Pi (Trading) Ltd.
 *
 * SPDX-License-Identifier: BSD-3-Clause
 */

#include <stdio.h>

#include <FreeRTOS.h>
#include <task.h>
#include <semphr.h>
#include <time.h>

#include "hardware/gpio.h"
#include "pico/stdlib.h"

const uint8_t LED_PIN = 25;
const uint8_t SW1_PIN = 17;
/* SW2 CREATED DURING PT 7 */
const uint8_t SW2_PIN = 16;

uint32_t heartbeatDelay = 500;  // ms

static SemaphoreHandle_t _semBtn1 = NULL;
static SemaphoreHandle_t _semBtn2 = NULL;
static SemaphoreHandle_t _semBoth = NULL;

volatile bool sw1_state = false;
volatile bool sw2_state = false;

volatile absolute_time_t last_interrupt;
volatile absolute_time_t current;

TaskHandle_t hb_task = NULL;
TaskHandle_t sw1_task = NULL;
TaskHandle_t sw2_task = NULL;

void gpio_int_callback(uint gpio, uint32_t events_unused) {
    printf("switch callback: GPIO ISR %d\n", gpio);

    if (gpio == SW1_PIN) {
        /* REMOVED AS PER PART 6 IN LAB 2 */
        //heartbeatDelay /= 2;
        xSemaphoreGiveFromISR(_semBtn1, NULL);
    }

    if (gpio == SW2_PIN) {
        xSemaphoreGiveFromISR(_semBtn2, NULL);
    }

    if (heartbeatDelay < 50) heartbeatDelay = 500;
}
```

```
void hardware_init(void) {
    const uint LED_PIN = PICO_DEFAULT_LED_PIN;
    gpio_init(LED_PIN);
    gpio_set_dir(LED_PIN, GPIO_OUT);

    gpio_init(SW1_PIN);
    gpio_pull_up(SW1_PIN);
    gpio_set_dir(SW1_PIN, GPIO_IN);
    gpio_set_irq_enabled_with_callback(SW1_PIN, GPIO_IRQ_EDGE_FALL, true,
&gpio_int_callback);

    gpio_init(SW2_PIN);
    gpio_pull_up(SW2_PIN);
    gpio_set_dir(SW2_PIN, GPIO_IN);
    gpio_set_irq_enabled_with_callback(SW2_PIN, GPIO_IRQ_EDGE_FALL, true,
&gpio_int_callback);

    last_interrupt = current = get_absolute_time();
}

/* Core Heartbeat Function.
   50% duty cycle.
   in ms                   */
void hb_core(uint32_t ms) {
    printf("hb-tick: %d\n", ms);
    gpio_put(LED_PIN, 1);
    vTaskDelay(ms);
    gpio_put(LED_PIN, 0);
    vTaskDelay(ms);
}

#if 1
void sw1_handler(void * notUsed) {
    while (true) {
        xSemaphoreTake( _semBtn1, portMAX_DELAY);
        printf("sw1 Semaphore taken..\n");
    }
}
#endif

/* INTERRUPT FOR SW2 CODE FOR PT 7 */
#if 1
void sw2_handler(void *notUsed) {
    while(true) {
        xSemaphoreTake(_semBtn2, portMAX_DELAY);
        printf("sw2 Semaphore taken...\n");
    }
}
#endif

/* MY NEW INTERRUPT HANDLERS
   PART 8 IN LAB 2          */
#if 1
```

```
void new_sw1_handler(void *notUsed) {
    while(true) {
        xSemaphoreTake(_semBtn1, portMAX_DELAY);
        gpio_set_irq_enabled_with_callback(SW1_PIN, GPIO_IRQ_EDGE_FALL, false,
&gpio_int_callback);
        last_interrupt = get_absolute_time();
        vTaskDelay(25);
        gpio_set_irq_enabled_with_callback(SW1_PIN, GPIO_IRQ_EDGE_FALL, true,
&gpio_int_callback);
        sw1_state = true;
        if (sw2_state) xSemaphoreGiveFromISR(_semBoth, NULL);
        uint32_t sw1_hb = 33;
        printf("sw2 Semaphore taken, hb set to %d\n", sw1_hb);
        for (int i = 0; i < 10; i++) {
            hb_core(sw1_hb);
        }
        sw1_state = false;
    }
}

void new_sw2_handler(void *notUsed) {
    while(true) {
        xSemaphoreTake(_semBtn2, portMAX_DELAY);
        gpio_set_irq_enabled_with_callback(SW2_PIN, GPIO_IRQ_EDGE_FALL, false,
&gpio_int_callback);
        last_interrupt = get_absolute_time();
        vTaskDelay(25);
        gpio_set_irq_enabled_with_callback(SW2_PIN, GPIO_IRQ_EDGE_FALL, true,
&gpio_int_callback);
        vTaskDelay(25);
        sw2_state = true;
        if (sw1_state) xSemaphoreGiveFromISR(_semBoth, NULL);
        uint32_t sw2_hb = 38;
        printf("sw2 Semaphore taken, hb set to %d\n", sw2_hb);
        for (int i = 0; i < 10; i++) {
            hb_core(sw2_hb);
        }
        sw2_state = false;
    }
}
#endif

void both_handler(void *notUsed) {
    uint32_t both_hb = 100;
    while(true) {
        xSemaphoreTake(_semBoth, portMAX_DELAY);
        last_interrupt = get_absolute_time();

        vTaskSuspend(hb_task);
        vTaskSuspend(sw1_task);
        vTaskSuspend(sw2_task);

        while(!gpio_get(SW1_PIN) && !gpio_get(SW2_PIN)) hb_core(both_hb);
        vTaskResume(hb_task);
        vTaskResume(sw1_task);
        vTaskResume(sw2_task);
    }
    sw2_state = false;
    sw2_state = false;
}

void remind() {
    printf("60s elapsed since last button press.\n");
    for(int i = 0; i < 50; i++) hb_core(25);
    last_interrupt = get_absolute_time();
    current = get_absolute_time();
}

void heartbeat(void * notUsed) {
    while (true) {
        current = get_absolute_time();
        if (current - last_interrupt > (60 * 1000000)) remind();
        hb_core(heartbeatDelay);
    }
}


int main() {
    stdio_init_all();
    printf("lab2 Hello!\n");
    hardware_init();

    _semBtn1 = xSemaphoreCreateBinary();
    _semBtn2 = xSemaphoreCreateBinary();
    _semBoth = xSemaphoreCreateBinary();


    xTaskCreate(heartbeat, "LED_Task", 256, NULL, 1, &hb_task);
    xTaskCreate(sw1_handler, "SW1_Task", 256, NULL, 2, NULL);
    xTaskCreate(sw2_handler, "SW2_Task", 256, NULL, 2, NULL);   // task created pt
7

    xTaskCreate(new_sw1_handler, "SW1_Task", 256, NULL, 3, &sw1_task);   // task
created pt 8
    xTaskCreate(new_sw2_handler, "SW2_Task", 256, NULL, 3, &sw2_task);   // task
created pt 8

    xTaskCreate(both_handler, "Both_Task", 256, NULL, 4, NULL); // Needed to finish
pt 9


    vTaskStartScheduler();

    while(1){};
}
```