Queues and Debugging

Tyler Petty

CS 566 Laboratory 3 Report

1 Objective

The purpose of this experiment was to introduce queues as well as debugging techniques by programming a simple producer consumer program while using software such as openocd and gdb to debug the program while it's running on the TI Launchpad.

2 Apparatus

Experimentation used the Texas Instruments Tiva C Series EK-TM4C123GXL Launchpad.

3 Method

Starting with the code from the second lab, the blinkyRTOS.c program was renamed to producerConsumer.c and had most of the code stripped out except for the _heartbeat() task and all the setup code for using the board. Then the make-file was modified by replacing any instance of blinkyRTOS with producerConsumer. Also, the make-file was checked to insure that it would include uartstdio.h and queue.h in the build. After that, some minor environment setup was needed including setting up the 99-platformio-udev.rules and adding the provided kermit executable to the path in order to prepare to receive UART prints from the board. Once this setup was done, the USB_SERIAL_OUTPUT was setup so that the board could send messages through the USB port to the serial communications program Kermit. At this point, the program was compiled and tested to ensure that everything was correctly set up to receive messages. The producer consumer portion of the project involved setting up two producer tasks and a single consumer task which would communicate using a 20 entry queue. The producer task was set up to block for a random delay between 50ms to 150ms to allow for an average of 10 messages produced per second. The consumer would then receive this message and turn on the corresponding LED, blue for producer 1 and red for producer 2. Each message consisted of a struct that includes the producers id number, an integer message, and a pointer to the queue which is used to set up the producers. Each producer is set up to assert if the message was able to be sent, and if it couldn't due to a queue overflow, the assert would alternate the LED from blue to red, block all other tasks, and print an error message to the serial communication console. Finally, a few exercises were performed using gdb and openocd, with openocd providing communications between gdb and the board.

4 Data

Found in the appendix is the code for producerConsumer.c. Below are some of the key points of the file.

4.1 Producer

The producer continuously sends a message to the queue, asserting that the queue is not full.

```
static void
_producer( void * pvParameters)
{
  uint32_t message = 1;
  struct producerData *prodData;
  prodData = (struct producerData *) pvParameters;
  while(1) {
    prodData->data = message;
    uint32_t delay = (rand() % (150 - 50 + 1)) + 50;
    vTaskDelay(delay / portTICK_RATE_MS);
    #ifdef USB_SERIAL_OUTPUT
      UARTprintf("Producing\n");
    #endif
    assert( xQueueSend( prodData->queue, prodData, 0) == pdPASS );
    message++;
  }
}
```

4.2 Consumer

The consumer continously checks the queue for an available message and will turn on one of the LEDs to indicate which producer sent the message

```
static void
_consumer( void * xQueue1 )
{
  struct producerData xRxed;
  while(1) {
    if( xQueueReceive( xQueue1,
                       &xRxed,
                       1 ) == pdPASS )
```

```
    {
      #ifdef USB_SERIAL_OUTPUT
        UARTprintf("Consumed %i from producer %i\n", xRxed.data, xRxed.producerId);
      #endif
      if ( xRxed.producerId == 1 ) {
        LED_ON(LED_B);
        vTaskDelay(50 / portTICK_RATE_MS);
        LED_OFF(LED_B);
      } else {
        LED_ON(LED_R);
        vTaskDelay(50 / portTICK_RATE_MS);
        LED_OFF(LED_R);
      }
    }
  }
}
```

5 Results and Analysis

FreeRTOS continues to be an easy to use environment for programming a Tiva Launchpad. The only issue encountered during the exercise was setting up the serial communication program Kermit, which continued to request "set line" even though it was set within the configuration file. Other than that, setting up the debugging environment with openocd and gdb was easy to do, though nothing could be found for the command "mr" stated in step 19 of the handout.

5.1 Questions

1, 2: How difficult was it to modify the code from lab 2 to the basic heartbeat-only task as well as renaming the main file
SW2 didn't have any visible effect with the default code, however SW1 changed the color of the led.
3: What does casting a value to be a (void *) do and what is is used for
Casting a value to be a (void *) tells the compiler to not associate a type to the address. This is generally used to suppress warnings if nothing is passed to the address.
Why does decreasing the priority of the consumer to below that of the producers cause an assert?
This is because the two producers are filling the queue faster than the consumer can empty it. With the producers having a higher priority, the scheduler will place them before the consumer whenever they are ready to run.
Do the other threads stop working after adding code from step 11 and why?
The additional code does cause the other threads to stop because taskENTER_CRITICAL and taskDISABLE_INTERRUPTS prevents the other tasks from being scheduled resources and interrupting the execution of the assert. In other words, the assert is given uninterrupted access to the boards resources and nothing else can be scheduled until the critical section is exited.
What happens when the above lines are commented out?
The other threads are still allowed to be scheduled and eventually the second producer fills the queue and asserts.
Display the first 16 32-bit values stored in ROM

```
(gdb) x /16xw 0x00000000
0x0 <g_pfnVectors>:        0x20007ab8        0x000034f9        0x0000358d        0x00003593
0x10 <g_pfnVectors+16>:        0x00003599        0x00003599        0x00003599        0x00000000
0x20 <g_pfnVectors+32>:        0x00000000        0x00000000        0x00000000        0x00000e31
0x30 <g_pfnVectors+48>:        0x00003599        0x00000000        0x00001081        0x000010e9
```

Display the processor registers

```
(gdb) info registers
r0             0x0                 0
r1             0xa5a5a5a5          -1515870811
r2             0xa5a5a5a5          -1515870811
r3             0xa5a5a5a5          -1515870811
r4             0xa5a5a5a5          -1515870811
r5             0xa5a5a5a5          -1515870811
r6             0xa5a5a5a5          -1515870811
r7             0x20000628          536872488
r8             0xa5a5a5a5          -1515870811
r9             0xa5a5a5a5          -1515870811
r10            0xa5a5a5a5          -1515870811
r11            0xa5a5a5a5          -1515870811
r12            0xa5a5a5a5          -1515870811
sp             0x20000628          0x20000628 <ucHeap+492>
lr             0xdcd               3533
```

```
pc             0x3148              0x3148 <_heartbeat+8>
xPSR           0x1000000           16777216
fpscr          0x0                 0
msp            0x20007a98          0x20007a98 <pui32Stack+4064>
psp            0x20000628          0x20000628 <ucHeap+492>
primask        0x0                 0
basepri        0x0                 0
faultmask      0x0                 0
control        0x2                 2
```

Display what breakpoints are set

```
(gdb) info breakpoints
Num     Type           Disp Enb Address    What
1       breakpoint     keep y   0x000031ce in main at producerConsumer.c:236
        breakpoint already hit 1 time
2       breakpoint     keep y   0x00003148 in _heartbeat at producerConsumer.c:191
        breakpoint already hit 1 time
```

What data is in the first 8 bytes of memory?
Assuming memory means SRAM.

```
(gdb) x /8xb 0x20000000
0x20000000 <xFreeBytesRemaining>:        0x38        0x51        0x00        0x00        0x00        0x
```

What do the following commands do?
(gdb) reload
(gdb) i b
(gdb) i r
(gdb) l
(gdb) mr
(gdb) s
(gdb) n

(gdb) reload
Restarts the current program in gdb
(gdb) i b
Prints out information of current breakpoints
(gdb) i r
Prints out information of the processor registers. For the tm4c123gxl board it prints registers R0 to control
(gdb) l
Prints 10 lines of code centered at the current or provided line.
(gdb) mr
Undefined command.
(gdb) s
Steps through code one line at a time.
(gdb) n
Steps through code like s but won't go into a new function.

6 Conclusion

The objectives of the experiment was to introduce FreeRTOS queues by programming a simple producer consumer program as well as introduce how to set up serial communication from the computer to the board through usb. The exersise also showed how to use openocd to allow the communication required for the debugging the program from the board using gdb debugger.

7 Appendix

7.1 producerConsumer.c

```
/* Standard includes. */
#include <stdbool.h>
#include <stdio.h>
#include <stdlib.h>

/* Kernel includes. */
#include "FreeRTOS.h"
#include "task.h"

/* Hardware includes. */
#include "inc/tm4c123gh6pm.h"
#include "driverlib/sysctl.h"
#include "driverlib/gpio.h"
#include "inc/hw_memmap.h"

/* Optional includes for USB serial output */
#ifdef USB_SERIAL_OUTPUT
#include "driverlib/rom.h"
#include "utils/uartstdio.h"
#include "driverlib/pin_map.h"
#include "driverlib/uart.h"
#endif

/*local includes*/
#include "assert.h"

// Included for GPIO_O_LOCK and others
#include "inc/hw_gpio.h" //for macro declaration of GPIO_O_LOCK and GPIO_O_CR
#include <inc/hw_types.h>

#include "timers.h"
#include "timing_verify.h"

#include "queue.h"

#define SPEED_TEST 0


#define LED_R (1<<1)
#define LED_G (1<<3)
#define LED_B (1<<2)
#define SW1   (1<<4)
#define SW2   (1<<0)

#define LED_ON(x) (GPIO_PORTF_DATA_R |= (x))
#define LED_OFF(x) (GPIO_PORTF_DATA_R &= ~(x))
#define LED(led,on) ((on)?LED_ON(led):LED_OFF(led))
#define pdTICKSTOMS( xTicks ) ((xTicks * 1000 ) / configTICK_RATE_HZ )

uint32_t SystemCoreClock;

#ifdef USB_SERIAL_OUTPUT

// #ifdef DEBUG
// void
// __error__(char *pcFilename, uint32_t ui32Line)
// {
//         _assert_failed ("__error__", pcFilename, ui32Line);
// }
// #endif

//*****************************************************************************
//
//: Configure the UART and its pins.  This must be called before UARTprintf().
//
//*****************************************************************************
static void
_configureUART(void)
```

```
{
    //
    // Enable UART0
    //
    SysCtlPeripheralEnable(SYSCTL_PERIPH_UART0);
    SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOA);
    //
    // Configure GPIO Pins for UART mode.
    //
    GPIOPinConfigure(GPIO_PA0_U0RX);
    GPIOPinConfigure(GPIO_PA1_U0TX);
    GPIOPinTypeUART(GPIO_PORTA_BASE, GPIO_PIN_0 | GPIO_PIN_1);

    //
    // Use the internal 16MHz oscillator as the UART clock source.
    //
    UARTClockSourceSet(UART0_BASE, UART_CLOCK_PIOSC);

    //
    // Initialize the UART for console I/O.
    //
    UARTStdioConfig(0, 115200, 16000000);
}

#endif

struct producerData {
  uint32_t data;
  uint32_t producerId;
  xQueueHandle queue;
};

static void
_setupHardware(void)
{
    //
    // Enable the GPIO port that is used for the on-board LED.
    // This is a TiveDriver library function
    //
    SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOF);
    HWREG(GPIO_PORTF_BASE + GPIO_O_LOCK) = GPIO_LOCK_KEY;
    HWREG(GPIO_PORTF_BASE + GPIO_O_CR) = 0x01;
    HWREG(GPIO_PORTF_BASE + GPIO_O_AFSEL) |= 0x01;

    // Enable the GPIO pin for the LED (PF3).  Set the direction as output, and
    // enable the GPIO pin for digital function.
    // These are TiveDriver library functions
    //
    GPIOPinTypeGPIOOutput(GPIO_PORTF_BASE, (LED_G|LED_R|LED_B));

    //
    // Set the clocking to run at (SYSDIV_2_5) 80.0 MHz from the PLL.
    //                            (SYSDIV_3) 66.6 MHz
    //                            (SYSDIV_4) 50.0 MHz
    //                            (SYSDIV_5) 40.0 MHz
    //                            (SYSDIV_6) 33.3 MHz
    //                            (SYSDIV_8) 25.0 MHz
    //                            (SYSDIV_10) 20.0 MHz
    //
    SystemCoreClock = 80000000;  // Required for FreeRTOS.

    SysCtlClockSet( SYSCTL_SYSDIV_2_5 |
                    SYSCTL_USE_PLL |
                    SYSCTL_XTAL_16MHZ |
                    SYSCTL_OSC_MAIN);

}

static void
_producer( void * pvParameters)
{
  uint32_t message = 1;
```

```
  struct producerData *prodData;
  prodData = (struct producerData *) pvParameters;
  while(1) {
    prodData->data = message;
    uint32_t delay = (rand() % (150 - 50 + 1)) + 50;
    vTaskDelay(delay / portTICK_RATE_MS);
    #ifdef USB_SERIAL_OUTPUT
      UARTprintf("Producing\n");
    #endif
    assert( xQueueSend( prodData->queue, prodData, 0) == pdPASS );
    message++;
  }
}

static void
_consumer( void * xQueue1 )
{
  struct producerData xRxed;
  while(1) {
    if( xQueueReceive( xQueue1,
                       &xRxed,
                       1 ) == pdPASS )
    {
      #ifdef USB_SERIAL_OUTPUT
        UARTprintf("Consumed %i from producer %i\n", xRxed.data, xRxed.producerId);
      #endif
      if ( xRxed.producerId == 1 ) {
        LED_ON(LED_B);
        vTaskDelay(50 / portTICK_RATE_MS);
        LED_OFF(LED_B);
      } else {
        LED_ON(LED_R);
        vTaskDelay(50 / portTICK_RATE_MS);
        LED_OFF(LED_R);
      }
    }
  }
}

static void
_heartbeat( void *notUsed )
{
    uint32_t green500ms = 500; // 1 second
    uint32_t ledOn = 0;

    TickType_t current_time = 0;
    TickType_t last_time = 0;

    while(true)
    {
        ledOn = !ledOn;
        if (ledOn) {
          current_time = pdTICKSTOMS(xTaskGetTickCount());
          timing_freq_datapoint(3, (current_time - last_time));
          last_time = current_time;
        }
        LED(LED_G, ledOn);

        vTaskDelay(green500ms / portTICK_RATE_MS);
    }
}

static void _timing_verification(void *notUsed) {
  while(true) {
    for (int j = 1; j <= 3; j++) {
      uint32_t timing = 0;
      // LED(LED_R, 0);
      // LED(LED_G, 0);
      // LED(LED_B, 0);
      if (j == 3)
        timing = timing_verify_frequency_in_range(3, 900, 1100);
      if (j == 2)
```

```
        timing = timing_verify_frequency_in_range(2, 70, 80);
      if (j == 1)
        timing = timing_verify_frequency_in_range(1, 0, 1100);
      if ( timing != 0 ) {
        LED(1<<j, 1);
      }
      vTaskDelay(500 / portTICK_RATE_MS);
    }
    vTaskDelay(1000 / portTICK_RATE_MS);
  }
}


int main( void )
{
    _setupHardware();

    #ifdef USB_SERIAL_OUTPUT
            void spinDelayMs(uint32_t ms);
            _configureUART();
            spinDelayMs(1000);  // Allow UART to setup
            UARTprintf("Hello from producerConsumer main()\n");
      UARTprintf("Hello, modification for step 18\n");
    #endif
    xTaskCreate(_heartbeat,
                "green",
                configMINIMAL_STACK_SIZE,
                NULL,
                tskIDLE_PRIORITY + 1,  // higher numbers are higher priority..
                NULL );

    xQueueHandle xQueue1;
    xQueue1 = xQueueCreate( 20, sizeof( struct producerData ) );
    if (xQueue1 == NULL) {
      #ifdef USB_SERIAL_OUTPUT
              UARTprintf("Queue failed to generate\n");
      #endif
    }

    struct producerData producer1data;
    producer1data.producerId = 1;
    producer1data.queue = xQueue1;

    struct producerData producer2data;
    producer2data.producerId = 2;
    producer2data.queue = xQueue1;

    uint32_t producerPriority = 2;
    uint32_t consumerPriority = 3;

    xTaskCreate(_producer,
                "producer1",
                configMINIMAL_STACK_SIZE,
                &producer1data,
                tskIDLE_PRIORITY + producerPriority,  // higher numbers are higher priority..
                NULL );

    xTaskCreate(_producer,
                "producer2",
                configMINIMAL_STACK_SIZE,
                &producer2data,
                tskIDLE_PRIORITY + producerPriority,  // higher numbers are higher priority..
                NULL );

    xTaskCreate(_consumer,
                "consumer",
                configMINIMAL_STACK_SIZE,
                ( void * ) xQueue1,
                tskIDLE_PRIORITY + consumerPriority,  // higher numbers are higher priority..
                NULL );

    /* Start the tasks and timer running. */\
```

```
    vTaskStartScheduler();

    assert(0); // we should never get here..

    return 0;
}
```