Nick Scheele

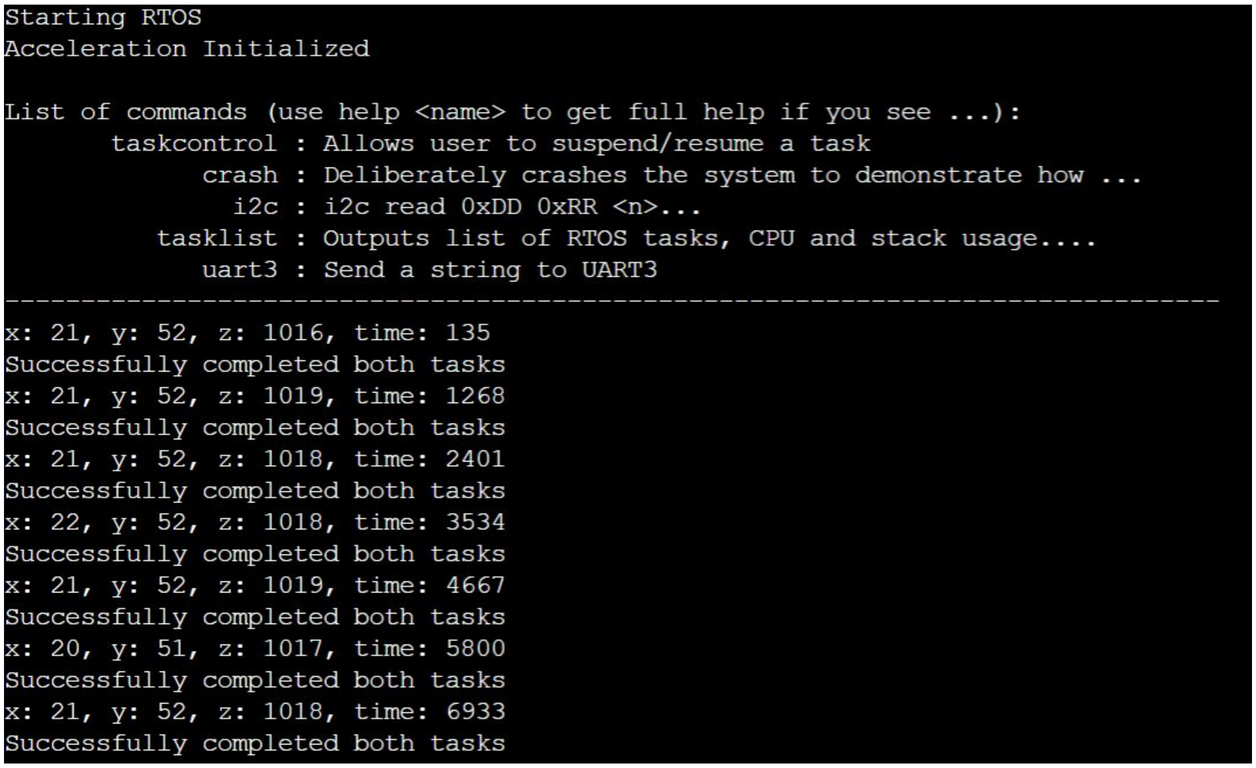
Prabjyot Obhi

Vivek Naveen

Watchdog Lab

Part 0 Code, Screenshot, and Explanation:

In part 0 of the lab, we had to create a producer and consumer task in which the data sent over the queue is the average value of 100 accelerometer readings. A function to get the average value was created, then, it was sent via the RTOS queue to be received by the consumer. Since the consumer has a portMAX_DELAY in the xQueueReceive, it will wait until the queue has a value to be received. Furthermore, the values were then printed to a file on an SD Card. The SD Card files have been submitted separately.



```
Starting RTOS
Acceleration Initialized

List of commands (use help <name> to get full help if you see ...):
       taskcontrol : Allows user to suspend/resume a task
             crash : Deliberately crashes the system to demonstrate how ...
               i2c : i2c read 0xDD 0xRR <n>...
          tasklist : Outputs list of RTOS tasks, CPU and stack usage....
             uart3 : Send a string to UART3
--------------------------------------------------------------------------------
x: 21, y: 52, z: 1016, time: 135
Successfully completed both tasks
x: 21, y: 52, z: 1019, time: 1268
Successfully completed both tasks
x: 21, y: 52, z: 1018, time: 2401
Successfully completed both tasks
x: 22, y: 52, z: 1018, time: 3534
Successfully completed both tasks
x: 21, y: 52, z: 1019, time: 4667
Successfully completed both tasks
x: 20, y: 51, z: 1017, time: 5800
Successfully completed both tasks
x: 21, y: 52, z: 1018, time: 6933
Successfully completed both tasks
```

main.c

```c
#include "acceleration.h"
#include "event_groups.h"
#include "ff.h"
#include <string.h>
int file_count = 0;
void write_file_using_fatfs_pi(acceleration__axis_data_s receive_value) {
  delay__ms(2);
  // file_count++;
  const char *filename = "file_plot.txt";
  FIL file; // File handle
  UINT bytes_written = 0;
  FRESULT result = f_open(&file, filename, (FA_WRITE | FA_OPEN_APPEND));

  if (FR_OK == result) {
    char string[64];
    // sprintf(string, "Value,%i\n", 123);
    sprintf(string, "x: %d, y: %d, z: %d\n", receive_value.x, receive_value.y,
receive_value.z);
    // fprintf(stderr, "printed\n");
    if (FR_OK == f_write(&file, string, strlen(string), &bytes_written)) {
    } else {
      printf("ERROR: Failed to write data to file\n");
    }
    f_close(&file);

  } else {
    printf("ERROR: Failed to open: %s\n", filename);
  }
}

static QueueHandle_t watchdog_queue;
static EventGroupHandle_t xCreateGroup;
acceleration__axis_data_s get_send_value() {
  acceleration__axis_data_s avg;
  int xVal = 0;
  int yVal = 0;
  int zVal = 0;
  for (int i = 0; i < 100; i++) {
    xVal += acceleration__get_data().x;
    yVal += acceleration__get_data().y;
    zVal += acceleration__get_data().z;
    delay__ms(1);
  }
  avg.x = xVal / 100;
  avg.y = yVal / 100;
  avg.z = zVal / 100;
  return avg;
}

void producer_of_sensor(void *p) {
  // fprintf(stderr, "producer\n");
  while (1) {
    // This xQueueSend() will internally switch context to "consumer" task because
it is higher priority than this
    // "producer" task Then, when the consumer task sleeps, we will resume out of
xQueueSend()and go over to the next
    // line
```

```c
    // TODO: Get some input value from your board
    acceleration__axis_data_s send_value = get_send_value();
    xQueueSend(watchdog_queue, &send_value, 0);
    xEventGroupSetBits(xCreateGroup, (1 << 1));
    vTaskDelay(1000);
  }
}

// TODO: Create this task at PRIORITY_HIGH
void consumer_of_sensor(void *p) {
  acceleration__axis_data_s receive_value;
  while (1) {
    if (xQueueReceive(watchdog_queue, &receive_value, portMAX_DELAY)) {
      uint32_t time = xTaskGetTickCount();
      fprintf(stderr, "x: %d, y: %d, z: %d, time: %d\n", receive_value.x,
receive_value.y, receive_value.z, time);
      write_file_using_fatfs_pi(receive_value);
    }
    xEventGroupSetBits(xCreateGroup, (1 << 2));
  }
}
```

Part 1 Code and Explanation:

In part 1 of the lab, the software watchdog was implemented. Here, we use the Event Group API given to us by FreeRTOS to set bit 1 when the producer task finishes the while loop and set bit 2 when the consumer finishes the while loop. This way, we are able to check if both tasks finished, hence the “watchdog” watches over and monitors our application. Then, the file was printed to when either task did not finish with an error message. The error file is submitted separately.

main.c

```c
#include "acceleration.h"
#include "event_groups.h"
#include "ff.h"
#include <string.h>

void write_file_using_fatfs_pi_error(int err) {
  delay__ms(2);
  // file_count++;
  const char *filename = "error.txt";
  FIL file; // File handle
  UINT bytes_written = 0;
  FRESULT result = f_open(&file, filename, (FA_WRITE | FA_OPEN_APPEND));

  if (FR_OK == result) {
    char string[64];
    // sprintf(string, "Value,%i\n", 123);
    // sprintf(string, "x: %d, y: %d, z: %d\n", receive_value.x, receive_value.y,
receive_value.z);
    if (err == 1) {
      sprintf(string, "Both tasks didn't finish\n");
    } else if (err == 2) {
      sprintf(string, "Consumer task didn't finish\n");
    }

    // fprintf(stderr, "printed\n");
    if (FR_OK == f_write(&file, string, strlen(string), &bytes_written)) {
    } else {
      printf("ERROR: Failed to write data to file\n");
    }
    f_close(&file);

  } else {
    printf("ERROR: Failed to open: %s\n", filename);
  }
}

void watchdog_task(void *params) {
  while (1) {
    vTaskDelay(1000);
    EventBits_t check = xEventGroupWaitBits(xCreateGroup, (1 << 1) | (1 << 2),
pdTRUE, pdFALSE, 200);
    if (((check & (1 << 1)) != 0) && ((check & (1 << 2)) != 0)) {
      fprintf(stderr, "Successfully completed both tasks\n");
    } else {
      // fprintf(stderr, "Producer task unfinished\n");
      int err;
      if (check & (1 << 1)) {
        fprintf(stderr, "Consumer task unfinished\n");
        err = 1;
        write_file_using_fatfs_pi_error(err);
      } else if (check & (1 << 2)) {
        fprintf(stderr, "Producer task unfinished\n");
        err = 2;
        write_file_using_fatfs_pi_error(err);
      } else {
        fprintf(stderr, "Both tasks didnt finish\n");
      }
    }
  }
}
```

Part 2 Code, Screenshots, and Explanations:

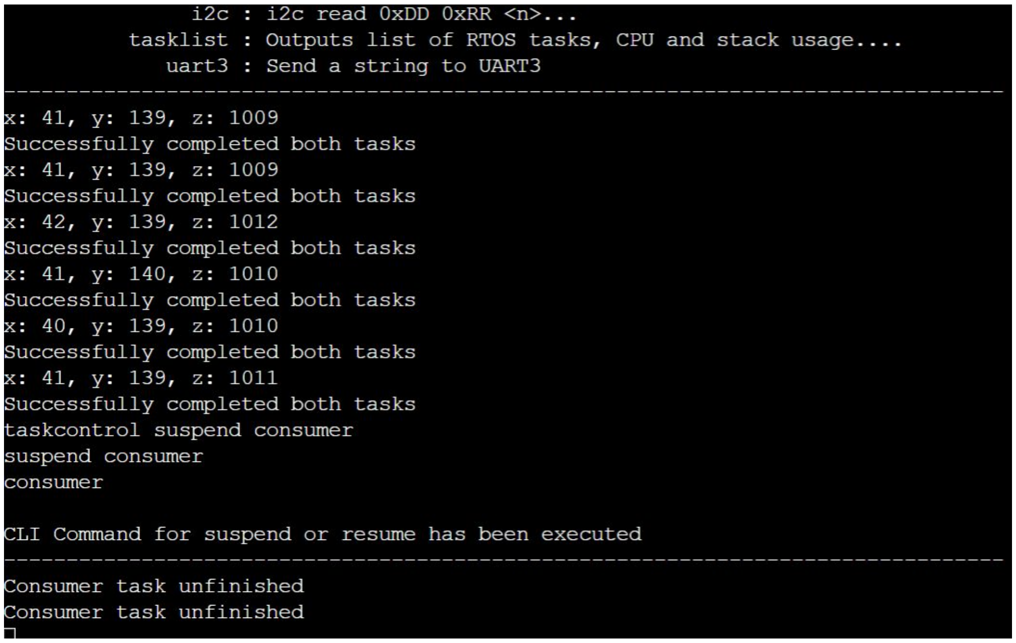
In part 2 of the lab, we used CLI commands that were written to suspend and resume tasks. These CLI commands were reused from last lab. When a consumer task is suspended, we expect for the bit at the end of the loop not to be set, and when this happens, we expect our error message to display on both the sd card as well as the telemetry output. When the consumer task is suspended, we get the message that the consumer task did not finish. Then, when the producer task is suspended, we get an error message that both tasks did not complete. This is because when the producer task is suspended, the consumer task will sleep forever as it has a portMAX_DELAY. Below these screenshots, we see the accelerometer values plotted on a line graph. The x-axis for this graph is the time, which found using the xTaskGetTickCount API in FreeRTOS. We can see from the plot that the accelerometers x-value is in a sinusoidal wave form, and this is because the board was flipped by the horizontal axis when taking data. The y and z values are relatively stable, and this is because the board was only flipped on the horizontal axis. The full source code is shown below.

main.c

```c
#include <stdio.h>

#include "FreeRTOS.h"
#include "task.h"
#include "board_io.h"
#include "common_macros.h"
#include "periodic_scheduler.h"
#include "queue.h"
#include "sj2_cli.h"
#include "acceleration.h"
#include "event_groups.h"
#include "ff.h"
#include <string.h>
int file_count = 0;
void write_file_using_fatfs_pi(acceleration__axis_data_s receive_value) {
  const char *filename = "WatchDog_Data.csv";
  FIL file; // File handle
  UINT bytes_written = 0;

  // Open for the first time
  FRESULT result = f_open(&file, filename, (FA_WRITE | FA_CREATE_NEW));
  if (FR_OK == result) {
    char string[64] = "";
    // sprintf(string, "Value,%i\n", 123);
    sprintf(string, "x,y,z\n");
    if (FR_OK == f_write(&file, string, strlen(string), &bytes_written)) {
    } else {
      printf("ERROR: Failed to write data to file\n");
    }
    sprintf(string, "%d,%d,%d\n", receive_value.x, receive_value.y,
receive_value.z);
    if (FR_OK == f_write(&file, string, strlen(string), &bytes_written)) {
    } else {
      printf("ERROR: Failed to write data to file\n");
    }
    f_close(&file);

  } else {
    result = f_open(&file, filename, (FA_WRITE | FA_OPEN_APPEND));
    if (FR_OK == result) {
      char string[64];
      // sprintf(string, "Value,%i\n", 123);
      sprintf(string, "%d,%d,%d\n", receive_value.x, receive_value.y,
receive_value.z);
      // fprintf(stderr, "printed\n");
      if (FR_OK == f_write(&file, string, strlen(string), &bytes_written)) {
      } else {
        printf("ERROR: Failed to write data to file\n");
      }
      f_close(&file);

    } else {
      printf("ERROR %d: Failed to open: %s\n", result, filename);
    }
  }
}

void write_file_using_fatfs_pi_error(int err) {
  delay__ms(2);
  // file_count++;
  const char *filename = "error.txt";
  FIL file; // File handle
  UINT bytes_written = 0;
  FRESULT result = f_open(&file, filename, (FA_WRITE | FA_OPEN_APPEND));

  if (FR_OK == result) {
    char string[64];
    // sprintf(string, "Value,%i\n", 123);
    // sprintf(string, "x: %d, y: %d, z: %d\n", receive_value.x, receive_value.y,
receive_value.z);
    if (err == 1) {
      sprintf(string, "Both tasks didn't finish\n");
    } else if (err == 2) {
      sprintf(string, "Consumer task didn't finish\n");
    }

    // fprintf(stderr, "printed\n");
    if (FR_OK == f_write(&file, string, strlen(string), &bytes_written)) {
    } else {
      printf("ERROR: Failed to write data to file\n");
    }
    f_close(&file);

  } else {
    printf("ERROR: Failed to open: %s\n", filename);
  }
}

static QueueHandle_t watchdog_queue;
static EventGroupHandle_t xCreateGroup;
acceleration__axis_data_s get_send_value() {
  acceleration__axis_data_s avg;
  int xVal = 0;
  int yVal = 0;
  int zVal = 0;
  for (int i = 0; i < 100; i++) {
    xVal += acceleration__get_data().x;
    yVal += acceleration__get_data().y;
    zVal += acceleration__get_data().z;
    delay__ms(1);
  }
  avg.x = xVal / 100;
  avg.y = yVal / 100;
  avg.z = zVal / 100;
  return avg;
}

void producer_of_sensor(void *p) {
  // fprintf(stderr, "producer\n");
  while (1) {
    // This xQueueSend() will internally switch context to "consumer" task because
it is higher priority than this
    // "producer" task Then, when the consumer task sleeps, we will resume out of
xQueueSend()and go over to the next
    // line

    // TODO: Get some input value from your board
    acceleration__axis_data_s send_value = get_send_value();

    // TODO: Print a message before xQueueSend()
    // Note: Use printf() and not fprintf(stderr, ...) because stderr is a polling
printf
    // fprintf(stderr, "before xqueuesend\n");
    // printf("send");
    xQueueSend(watchdog_queue, &send_value, 0);
    // TODO: Print a message after xQueueSend()
    // fprintf(stderr, "after xqueuesend\n");
    // fprintf(stderr, "------------------producer done--------------");
    xEventGroupSetBits(xCreateGroup, (1 << 1));
```

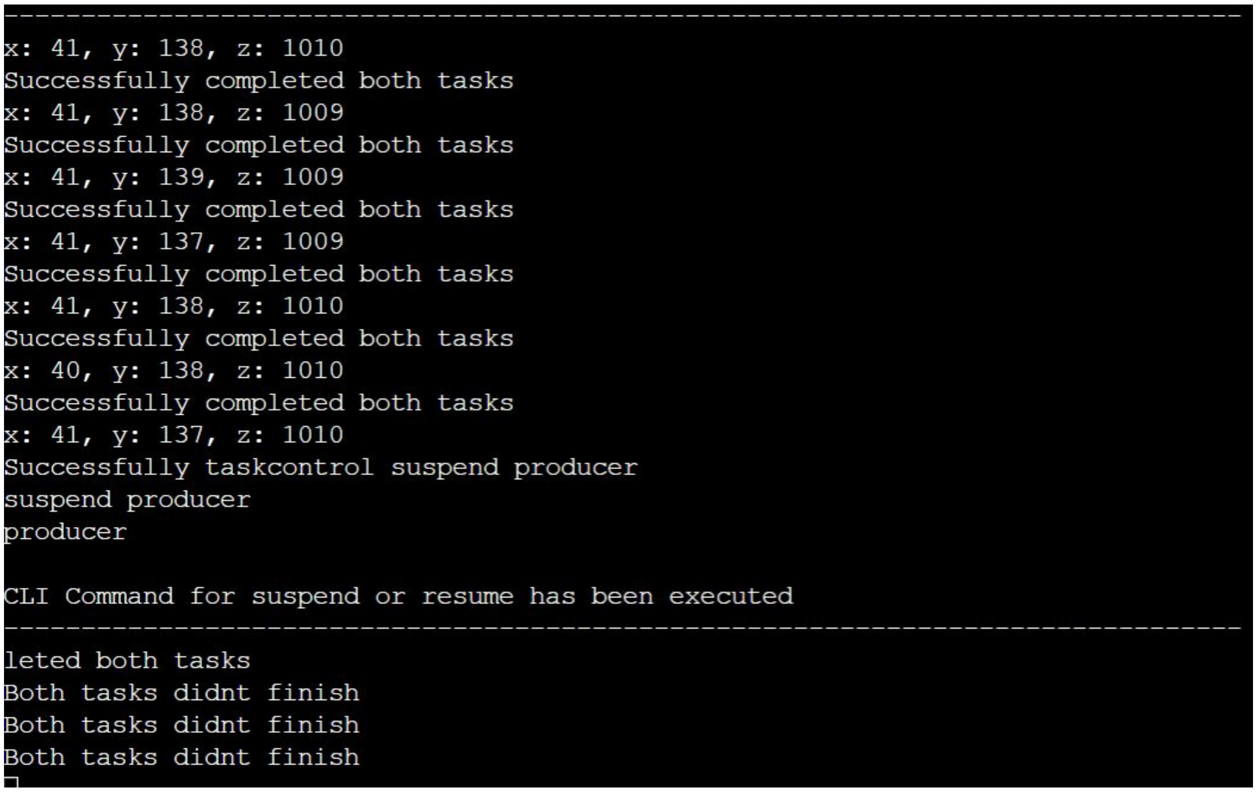
```c
    vTaskDelay(1000);
  }
}

// TODO: Create this task at PRIORITY_HIGH
void consumer_of_sensor(void *p) {
  acceleration__axis_data_s receive_value;
  while (1) {
    // TODO: Print a message before xQueueReceive()
    // fprintf(stderr, "before xqueuereceive\n");
    if (xQueueReceive(watchdog_queue, &receive_value, portMAX_DELAY)) {
      uint32_t time = xTaskGetTickCount();
      fprintf(stderr, "x: %d, y: %d, z: %d, time: %d\n", receive_value.x,
receive_value.y, receive_value.z, time);
      write_file_using_fatfs_pi(receive_value);
    }
    xEventGroupSetBits(xCreateGroup, (1 << 2));
  }
}

void watchdog_task(void *params) {
  while (1) {
    vTaskDelay(1000);
    // if (xEventGroupWaitBits(xCreateGroup, (1 << 1), pdTRUE, pdFALSE, 100) &&
    //     xEventGroupWaitBits(xCreateGroup, (1 << 2), pdTRUE, pdFALSE, 100)) {
    //   fprintf(stderr, "Successfully completed both tasks\n");
    // } else {
    //   if (!xEventGroupWaitBits(xCreateGroup, (1 << 1), pdTRUE, pdFALSE, 0)) {
    //     fprintf(stderr, "Producer task unfinished\n");
    //   }
    //   if (!xEventGroupWaitBits(xCreateGroup, (1 << 2), pdTRUE, pdFALSE, 0)) {
    //     fprintf(stderr, "Consumer task unfinished\n");
    //   }
    // }

    // if (xEventGroupWaitBits(xCreateGroup, (1 << 1) | (1 << 2), pdTRUE, pdFALSE,
200)) {
    //   fprintf(stderr, "Successfully completed both tasks\n");
    // } else {
    //   // fprintf(stderr, "Producer task unfinished\n");
    //   if (xEventGroupWaitBits(xCreateGroup, (1 << 1), pdTRUE, pdFALSE, 0)) {
    //     fprintf(stderr, "Consumer task unfinished\n");
    //   } else if (xEventGroupWaitBits(xCreateGroup, (1 << 2), pdTRUE, pdFALSE,
0)) {
    //     fprintf(stderr, "Producer task unfinished\n");
    //   } else {
    //     fprintf(stderr, "Both tasks didnt finish\n");
    //   }
    // }
    EventBits_t check = xEventGroupWaitBits(xCreateGroup, (1 << 1) | (1 << 2),
pdTRUE, pdFALSE, 200);
    if (((check & (1 << 1)) != 0) && ((check & (1 << 2)) != 0)) {
      fprintf(stderr, "Successfully completed both tasks\n");
    } else {
      // fprintf(stderr, "Producer task unfinished\n");
      int err;
      if (check & (1 << 1)) {
        fprintf(stderr, "Consumer task unfinished\n");
        err = 1;
        write_file_using_fatfs_pi_error(err);
      } else if (check & (1 << 2)) {
        fprintf(stderr, "Producer task unfinished\n");
        err = 2;
        write_file_using_fatfs_pi_error(err);
      } else {
        fprintf(stderr, "Both tasks didnt finish\n");
      }
    }
  }
}
void watchdog_main() {
  if (acceleration__init()) {
    fprintf(stderr, "Acceleration Initialized\n");
  }
  TaskHandle_t prod_watchdog;
  TaskHandle_t cons_watchdog;
  TaskHandle_t watchdog;
  xTaskCreate(producer_of_sensor, "producer", 1024, NULL, 2, &prod_watchdog);
  xTaskCreate(consumer_of_sensor, "consumer", 1024, NULL, 2, &cons_watchdog);
  xTaskCreate(watchdog_task, "watchdog", 1024, NULL, 3, &watchdog);
  // TODO Queue handle is not valid until you create it
  watchdog_queue = xQueueCreate(
      1, sizeof(acceleration__axis_data_s)); // Choose depth of item being our enum
(1 should be okay for this example
  xCreateGroup = xEventGroupCreate();
}

int main(void) {
  create_blinky_tasks();
  create_uart_task();

  // If you have the ESP32 wifi module soldered on the board, you can try
uncommenting this code
  // See esp32/README.md for more details
  // uart3_init();
// Also include:  uart3_init.h
```

```
  // xTaskCreate(esp32_tcp_hello_world_task, "uart3", 1000, NULL, PRIORITY_LOW,
NULL); // Include esp32_task.h

  puts("Starting RTOS");
  // producer_consumer_assignment();
  watchdog_main();
  vTaskStartScheduler(); // This function never returns unless RTOS scheduler runs
out of memory and fails

  return 0;
}
```



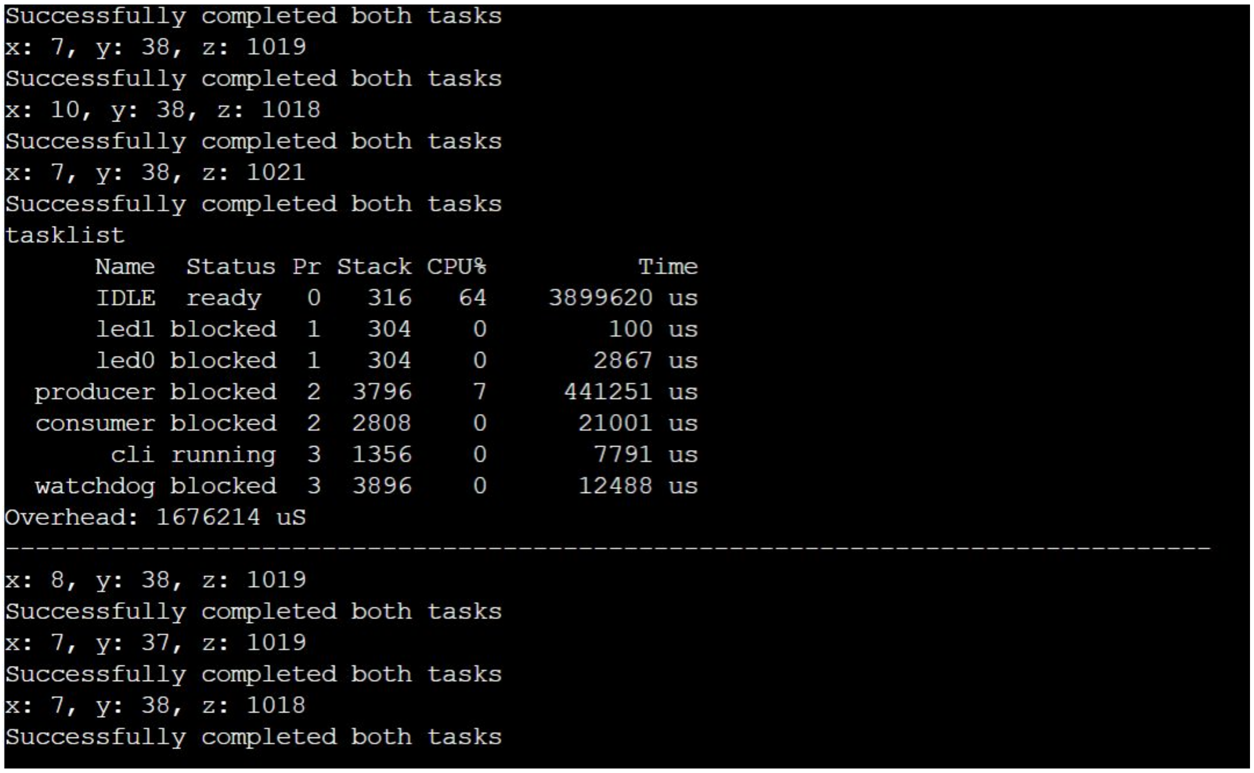
```
              i2c : i2c read 0xDD 0xRR <n>...
         tasklist : Outputs list of RTOS tasks, CPU and stack usage....
            uart3 : Send a string to UART3
--------------------------------------------------------------------------------
x: 41, y: 139, z: 1009
Successfully completed both tasks
x: 41, y: 139, z: 1009
Successfully completed both tasks
x: 42, y: 139, z: 1012
Successfully completed both tasks
x: 41, y: 140, z: 1010
Successfully completed both tasks
x: 40, y: 139, z: 1010
Successfully completed both tasks
x: 41, y: 139, z: 1011
Successfully completed both tasks
taskcontrol suspend consumer
suspend consumer
consumer

CLI Command for suspend or resume has been executed
--------------------------------------------------------------------------------
Consumer task unfinished
Consumer task unfinished
```

Consumer task is suspended

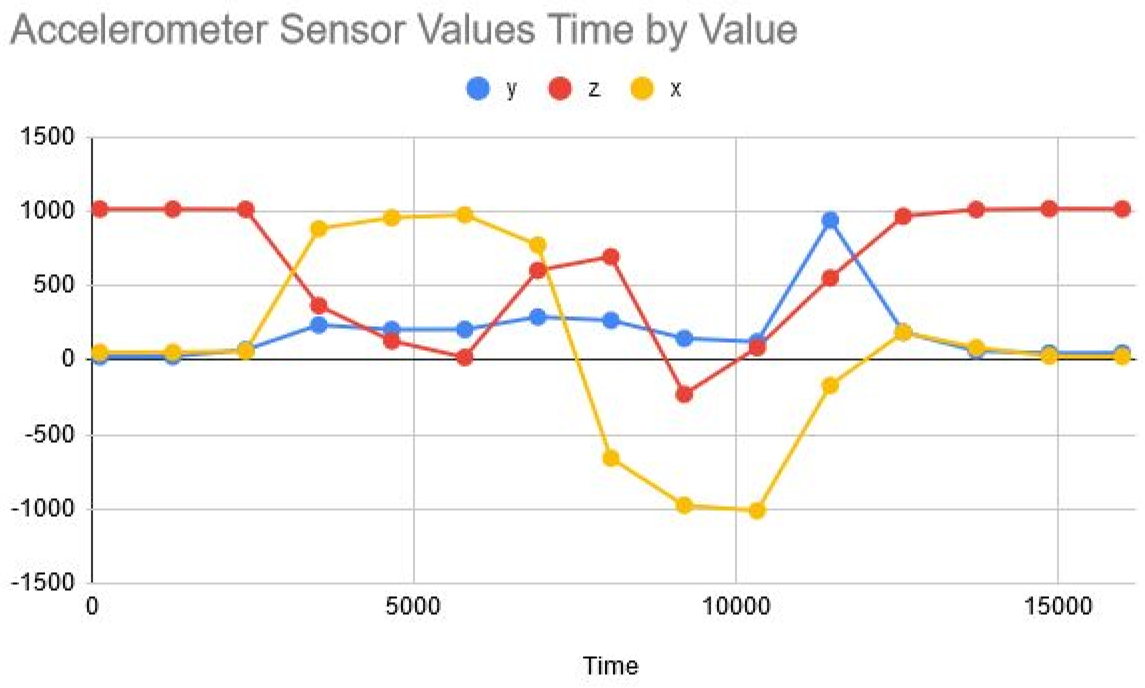


```
--------------------------------------------------------------------------------
x: 41, y: 138, z: 1010
Successfully completed both tasks
x: 41, y: 138, z: 1009
Successfully completed both tasks
x: 41, y: 139, z: 1009
Successfully completed both tasks
x: 41, y: 137, z: 1009
Successfully completed both tasks
x: 41, y: 138, z: 1010
Successfully completed both tasks
x: 40, y: 138, z: 1010
Successfully completed both tasks
x: 41, y: 137, z: 1010
Successfully taskcontrol suspend producer
suspend producer
producer

CLI Command for suspend or resume has been executed
--------------------------------------------------------------------------------
leted both tasks
Both tasks didnt finish
Both tasks didnt finish
Both tasks didnt finish
```

Producer task is suspended



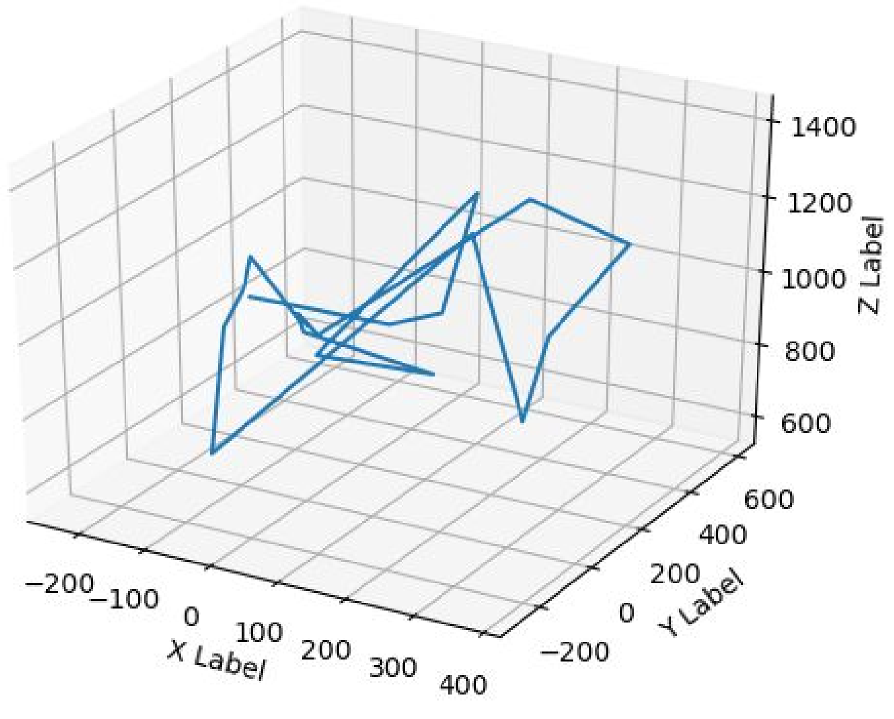
```
Successfully completed both tasks
x: 7, y: 38, z: 1019
Successfully completed both tasks
x: 10, y: 38, z: 1018
Successfully completed both tasks
x: 7, y: 38, z: 1021
Successfully completed both tasks
tasklist
      Name  Status Pr Stack CPU%          Time
      IDLE  ready   0   316   64    3899620 us
      led1 blocked  1   304    0        100 us
      led0 blocked  1   304    0       2867 us
  producer blocked  2  3796    7     441251 us
  consumer blocked  2  2808    0      21001 us
       cli running  3  1356    0       7791 us
  watchdog blocked  3  3896    0      12488 us
Overhead: 1676214 uS
-------------------------------------------------------------------------------
x: 8, y: 38, z: 1019
Successfully completed both tasks
x: 7, y: 37, z: 1019
Successfully completed both tasks
x: 7, y: 38, z: 1018
Successfully completed both tasks
```



Plotted values for accelerometer

3dGraph.py

```python
import matplotlib.pyplot as plt
import pandas as pd
from mpl_toolkits.mplot3d import Axes3D

print('Generate 3D Graph')
df = pd.read_csv('WatchDog_Data.csv')
df = df.dropna()

x = df['x'].to_numpy()
y = df['y'].to_numpy()
z = df['z'].to_numpy()

fig = plt.figure()
ax = fig.add_subplot(111, projection='3d')

ax.plot(x, y, z)

ax.set_xlabel('X Label')
ax.set_ylabel('Y Label')
ax.set_zlabel('Z Label')

plt.show()
```



3D Representation of the Accelerometer Readings