

Proyecto 2.

Alejandro Medina

Christian Mora

Marco Loaiza

Sistemas embebidos



Pontificia Universidad Javeriana
Facultad de Ingeniería
Abril 30 2021

1. Abstract

En el diseño de los sistemas embebidos, la programación de las tareas específicas que ejecutará el dispositivo es una parte vital para el correcto desarrollo de las soluciones, como una forma óptima de implementar tareas, se introduce un modelo particular de ejecución de programas llamado “threading” que permite correr varias funciones o tareas de forma no secuencial en un mismo procesador, emulando una ejecución en paralelo de varias funciones, no obstante, sí siendo los hilos secuenciales. Esto anterior permite un uso eficiente de los recursos del dispositivo a nivel de procesamiento, evitando que el sistema se quede congelado”desperdiciando tiempo sin ejecución, causado por un proceso que está a la espera de un dato o tarea ejecutable para seguir funcionando. En el presente documento del proyecto, se presenta un desarrollo que busca mostrar las ventajas que presenta el correcto funcionamiento del uso de hilos, para la óptima implementación de un sistema embebido.

In the design of embedded systems, programming specific tasks for the OS to execute, is a vital part for the correct development for solutions, as an optimal way to implement tasks, a particular model of program execution called ”threading”is introduced. ”This allows executing several functions or tasks (the execution of the program) non-sequentially on the same processor, emulating a parallel execution of several functions, however, the threads are sequential tasks. This allows efficient use of device resources at the processing level, preventing the system from becoming ”frozen”, wasting time without execution, caused by a process that is waiting for a data or executable task to continue running. In this project document, a development is presented that seeks to show the advantages of the correct operation of the use of threads, for the optimal implementation of an embedded system.

2. Introducción

Para poder verificar y entender el manejo que hace el sistema operativo con programas ejecutados en segundo plano o programas multitarea el proyecto propone dos objetivos principales:

1. Verificar el funcionamiento de dos scripts ejecutados en segundo plano.
2. Ejecutar un script multitarea mediante el uso de threads o hilos.

Para el desarrollo del primer objetivo se requiere:

- Escribir un programa en Co C++ que cambie el estado de un puerto definido a una frecuencia de 200KHz.
- Escribir un segundo programa, que cambie el estado de otro puerto del dispositivo con la misma frecuencia de 200KHz.
- Verificar el funcionamiento del sistema operativo con la ejecución de dos programas en segundo plano.

Para el desarrollo del segundo objetivo se requiere:

- Ejecutar un script en python que capture los datos de un sensor con comunicación I2C y los imprima en pantalla a una tasa de una muestra por segundo.
- Capturar los datos de un sensor 1 Wire y guardarlos en un archivo como se realizó en el proyecto 1.
- Implementar una aplicación en python utilizando hilos para ejecutar diferentes tareas.

Las tareas que se proponen son:

- Leer el sensor I2C a la máxima tasa de transferencia que permita el dispositivo.
- Leer el sensor 1-wire como se ejecutó en el proyecto1.
- Recibir por puerto serial una trama que indica la ventana de datos a utilizar para sacar el valor promedio de datos del sensor I2C.
- Escribir en el puerto serial el promedio de las últimas N lecturas del sensor I2C, indicadas en el punto anterior.

3. Arquitectura de Bloques General

Para el desarrollo del primer objetivo se realiza la configuración del puerto de la raspberry y se conecta a una resistencia junto a un led. Utilizando la librería de wiring pi se ejecuta la conmutación o subida y bajada del puerto con su respectivo delay función también de la librería que permite configurar el tiempo a la frecuencia requerida.



Figura 1: Diagrama de bloques blink.c .

Para la ejecución de los programas en segundo plano se propone el siguiente diagrama en bloques del script a ejecutar.

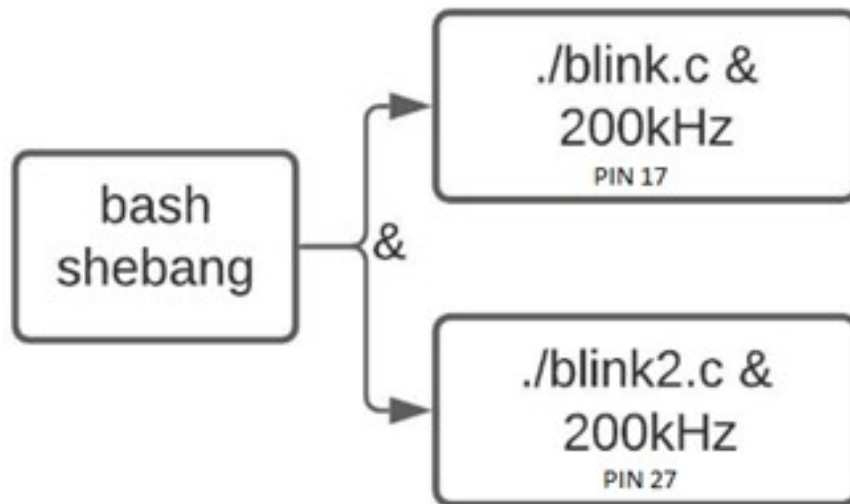


Figura 2: Diagrama de bloques programa bash.

Para el desarrollo del segundo objetivo se propone el siguiente diagrama en bloques.

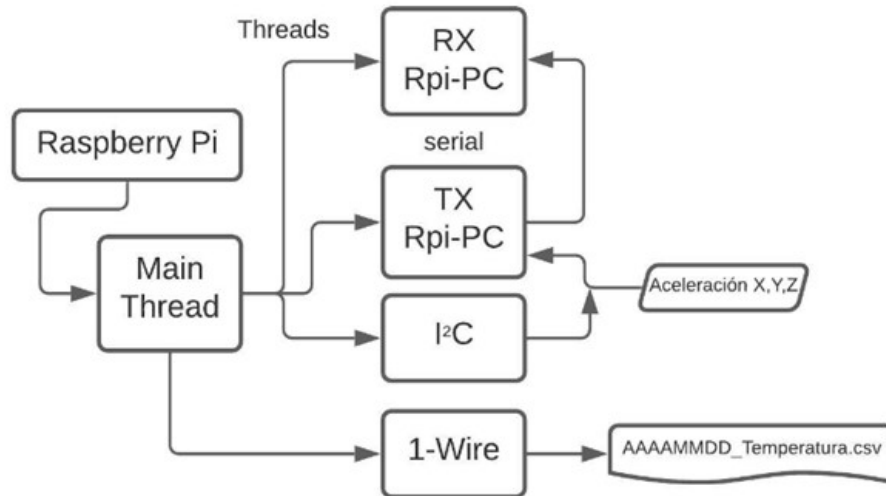


Figura 3: Diagrama de bloques programación.

Se implementan 5 hilos para manejar las diferentes tareas, el hilo RX encargado de leer el puerto serial cuando se envié la trama con formato: (“##*PROMEDIO*-NNN-##”).

El hilo TX el cual envía la información del promedio de los últimos N datos especificados por el numero N obtenido de la trama.

EL hilo 1-wire encargado de leer la temperatura del sensor 1-wire y guardarlo en el archivo con formato AAMMDD Temperatura.csv

El hilo I2C encargado de capturar la información del acelerómetro y ejecutar el promedio según los requerimientos descritos.

El hilo principal donde se realiza la configuración inicial del puerto serial y creación de los threads.

4. Desarrollo de la Solución

4.1. Primer objetivo

En el desarrollo del primer objetivo se toma como referencia el script ejecutado en el primer proyecto, se utiliza la librería wiring pi para su ejecución junto con la función udelay de la misma librería para cumplir con el requerimiento de conmutación de 200KHz.

```
#include <wiringPi.h>

const int PIN = 17;
const float Del = 2.5;

int main (void)
{
    wiringPiSetupGpio() ;
    pinMode (PIN, OUTPUT) ;
    for (;;)
    {
        digitalWrite (PIN, HIGH) ; udelay (Del) ;
        digitalWrite (PIN, LOW) ; udelay (Del) ;
    }
    return 0 ;
}
```

Figura 4: Código blink.c

Para una frecuencia de 200KHz el periodo de la señal corresponde a 5us el periodo corresponde a un ciclo completo de subida y bajada del puerto por lo cual el tiempo de espera es igual a la mitad del periodo de la señal obteniendo de esta manera un delay de 2.5us para cumplir con el objetivo.

Del proyecto 1 se conoce que el tiempo de conmutación de la librería wiringPi en C / C++ es de aproximadamente 13.72 Mhz que comparada con el valor de frecuencia de 200KHz no afecta en mayor medida para los cálculos del delay.

```
#!/bin/bash
./blink.c &
./blink2.c &
```

Figura 5: Código *blink.c*

Para la ejecución de los dos comandos se realiza mediante el script de bash que hace el llamado a los binarios de cada código compilado y las ejecuta en segundo plano.

4.2. Segundo objetivo

Para el desarrollo del segundo objetivo se ejecutan las diferentes tareas por aparte para garantizar su correcto funcionamiento y posteriormente se procede a programar el script con el uso de hilos.

Primero se ejecuta un programa para capturar los valores del sensor I2C utilizando Python a una tasa de una muestra por segundo.

```
import smbus
import time

while 1:
    bus = smbus.SMBus(1)
    bus.write_byte_data(0x53, 0x2C, 0x0A)
    bus.write_byte_data(0x53, 0x2D, 0x08)
    bus.write_byte_data(0x53, 0x31, 0x08)
    time.sleep(0.5)
    data0 = bus.read_byte_data(0x53, 0x32)
    data1 = bus.read_byte_data(0x53, 0x33)
    xAccl = ((data1 & 0x03) * 256) + data0
    if xAccl > 511 :
        xAccl -= 1024
    data0 = bus.read_byte_data(0x53, 0x34)
    data1 = bus.read_byte_data(0x53, 0x35)

    yAccl = ((data1 & 0x03) * 256) + data0
    if yAccl > 511 :
        yAccl -= 1024
    data0 = bus.read_byte_data(0x53, 0x36)
    data1 = bus.read_byte_data(0x53, 0x37)

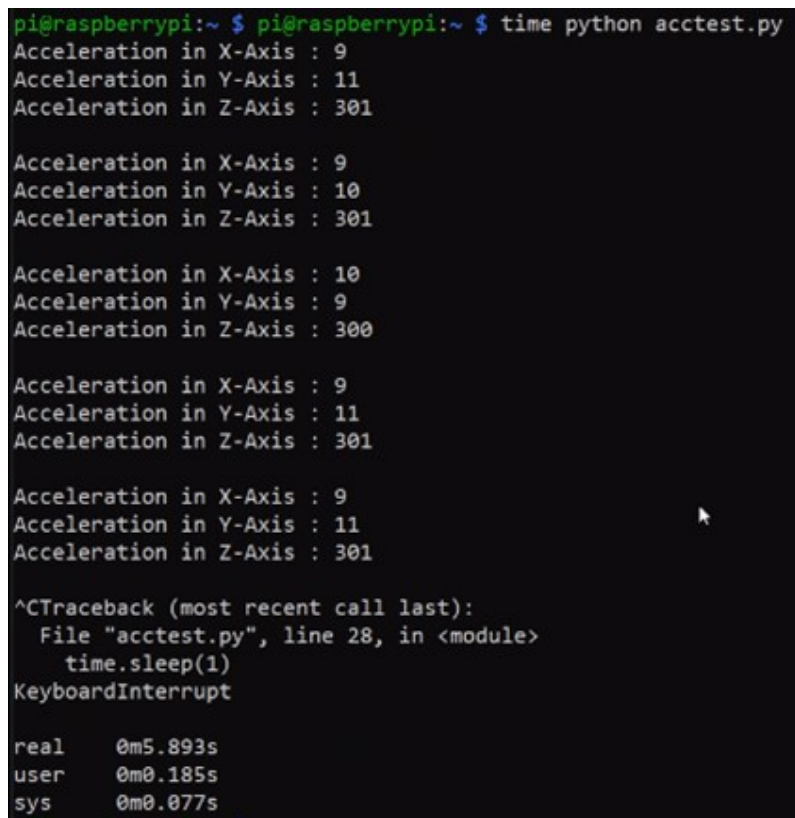
    zAccl = ((data1 & 0x03) * 256) + data0
    if zAccl > 511 :
        zAccl -= 1024

    print "Acceleration in X-Axis : %d" %xAccl
    print "Acceleration in Y-Axis : %d" %yAccl
    print "Acceleration in Z-Axis : %d" %zAccl
    time.sleep(1)
```

Figura 6: Código lectura de acelerometro.

En este código, con el comando `bus.write_byte_data()`, se establecen (escriben) las siguientes definiciones para el protocolo I2C para interpretar el sensor de aceleración. Primero el address del sensor ADLX345, la tasa del ancho de banda del registro y el modo de operación, en la primera línea con las entradas respectivamente 0x53(83), 0x2C(44) y 0x0A(10). Seguido se define el registro de control de alimentación, y si estará deshabilitada el auto sleep, con 0x2D(45) y 0x08(08). Finalmente se establece el formato de los datos y self test, con 0x31(49) y 0x08(08). Ahora para la lectura de datos, se lee desde 0x32(50), 2 bytes del eje X en LSB (0x32) y el MSB (0x33). Luego esta data en 2 bytes del eje X se convierte a 10 bits en la ecuación y condicional seguido. Lo mismo se realiza para los ejes Y y Z. Finalmente los datos recogidos de las aceleraciones en X, Y y Z, se entregan en las variables `xAccl`, `yAccl` y `zAccl`.

A continuación, se muestra la ejecución del programa mostrando los valores de aceleración de los diferentes ejes.



```
pi@raspberrypi:~ $ pi@raspberrypi:~ $ time python acctest.py
Acceleration in X-Axis : 9
Acceleration in Y-Axis : 11
Acceleration in Z-Axis : 301

Acceleration in X-Axis : 9
Acceleration in Y-Axis : 10
Acceleration in Z-Axis : 301

Acceleration in X-Axis : 10
Acceleration in Y-Axis : 9
Acceleration in Z-Axis : 300

Acceleration in X-Axis : 9
Acceleration in Y-Axis : 11
Acceleration in Z-Axis : 301

Acceleration in X-Axis : 9
Acceleration in Y-Axis : 11
Acceleration in Z-Axis : 301

^CTraceback (most recent call last):
  File "acctest.py", line 28, in <module>
    time.sleep(1)
KeyboardInterrupt

real    0m5.893s
user    0m0.185s
sys     0m0.077s
```

Figura 7: Ejecución de programa de lectura de acelerómetro.

En segundo lugar, se realiza la conexión del sensor 1-wire y se prueba el funcionamiento del script ejecutado en el proyecto1.


```

Temp.sh x
1  #!/bin/bash
2
3  init=$(date +%Y%m%d\ %H:%M:%S)"_ "$Temp"°C"
4  echo "Fecha Hora;Temperatura" > /home/pi/"${init}.csv"
5  while true :
6  do
7      Temp=$(cat /sys/bus/w1/devices/w1_bus_master1/28-3c01d075bc1a/temperature)
8      Temp=$(echo "scale=2; $Temp/1000" | bc -l)
9      echo $(date +%Y%m%d\ %H:%M:%S);"$Temp"°C" >> /home/pi/"${init}.csv"
10
11     sleep 10
12 done

```

Figura 8: Script Temperatura en bash.

Posterior a esto se procede a realizar y probar la configuración serial de la Raspberry pi al PC, para enviar y recibir datos.

```

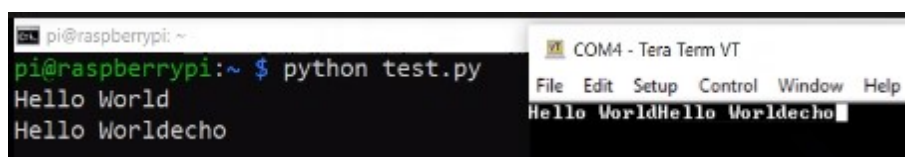
import time
import serial

ser = serial.Serial(
    port='/dev/ttyUSB0',
    baudrate = 9600,
    parity=serial.PARITY_NONE,
    stopbits=serial.STOPBITS_ONE,
    bytesize=serial.EIGHTBITS,
    timeout=1
)

counter=0
while 1:
    x=ser.readline()
    print x

```

Figura 9: Script prueba del puerto serial.



```

pi@raspberrypi: ~
pi@raspberrypi:~ $ python test.py
Hello World
Hello Worlddecho

```

```

COM4 - Tera Term VT
File Edit Setup Control Window Help
Hello WorldHello Worlddecho

```

Figura 10: Prueba del puerto serial.

Como se puede observar en la figura 10, se utiliza el programa `tera Term` como aplicación serial y poder visualizar de esta manera el envío y recepción de datos entre la raspberry y el computador.

Finalmente se diseñan las funciones de las diferentes tareas que se ejecutarán como hilos del programa mostradas por segmentos a continuación:

```
import time, serial, queue, threading
import statistics
import pt2_i2c as I2C
import subprocess

ser = None
RxTx = queue.Queue()
xI2CTx = queue.LifoQueue()
yI2CTx = queue.LifoQueue()
zI2CTx = queue.LifoQueue()

def TempThread():
    while 1:
        subprocess.call("./tempbash.sh")
        time.sleep(1)
```

Figura 11: Importación de librerías, declaración de variables y definición de función para el hilo de datos de temperatura.

Se importan las librerías, `time` para poder accionar la ejecución de los distintos threads con el comando `time.sleep()`, `serial` para poder implementar la comunicación serial a través de UART (Raspberry Pi - PC), `queue` para poder operar con la estructura LIFO los datos que llegan de las tres aceleraciones en X, Y y Z, `threading` para definir los 4 hilos de operación, los cuales son para el manejo del sensor I2C, otro para el sensor de temperatura por 1-Wire y los dos últimos para el RX y TX del serial, `statistics` para realizar el promedio de los datos de aceleración de cada eje. Se importaron las funciones de captura de datos de i2C y 1-Wire de dos archivos externos (`pt2_i2c` y `subprocess` respectivamente).

```

def mainThread(comPortName):
    global ser
    ser = serial.Serial \
    (
        port=comPortName,
        baudrate=115200,
        parity=serial.PARITY_NONE,
        stopbits=serial.STOPBITS_ONE,
        bytesize=serial.EIGHTBITS
    )

    threading.Thread(target=TempThread).start()
    threading.Thread(target=I2CThread).start()
    threading.Thread(target=RxThread).start()
    threading.Thread(target=TxThread).start()

    try:
        while 1:
            time.sleep(1)
    except:
        ser = None
if __name__ == "__main__":
    mainThread("/dev/ttyAMA0")

```

Figura 12: Definición de función para hilo principal.

El programa se inicia con la función `mainThread()` donde tiene como argumento de entrada el archivo del puerto serial que hará la comunicación serial, aquí se inicializan igualmente los parámetros de la comunicación serial y se definen e inician cada uno de los 4 threads, que ejecutan funciones donde se encuentran: la toma de datos de temperatura -`TempThread()`-, la toma de datos de aceleración -`I2CThread()`-, la recepción de datos por RX -`RxThread()`- y la transmisión de datos por TX -`TxThread()`- luego se mantiene la ejecución del `mainThread()` utilizando el argumento Try-except wut en dado caso que su ejecución falle terminará la ejecución una vez complete el hilo.

```

def I2CThread():
    while 1:
        I2C.i2cRTX()
        xI2CTx.put(I2C.xAcc1)
        yI2CTx.put(I2C.yAcc1)
        zI2CTx.put(I2C.zAcc1)

def RxThread():
    while 1:
        x = '#'

        if ser.inWaiting() > 0:
            while ser.inWaiting() > 0:
                aux = ser.read()
                x = x + aux
            RxTx.put(x)
            print(x)
        else:
            time.sleep(0.2)

```

Figura 13: Definición de funciones I2C y recepción de datos

El hilo I2CThread llama la función I2C.i2cRTX() para capturar la información del sensor por cada uno de los ejes y los asigna a la cola de datos.

El hilo RxThread() se encarga de recibir la trama almacenarla en una variable y agregarla a la cola RxTx cada vez que identifica una trama llegando por el puerto serial.

Se inicializa la variable x para evitar sobre posición de datos anteriores con las nuevas tramas recibidas.

```

def TxThread():
    Datos = ['###PROMEDIO', '010', '##\n']
    auxN = int(Datos[1])

    while 1:
        x, y, z = [], [], []
        if RxTx.empty():
            time.sleep(0.2)
        else:
            N = RxTx.get()
            Datos = N.split('-')
            if str(Datos[0]) == '###PROMEDIO' and str(Datos[2]) == ('##' + '\n'):
                if xI2CTx.empty():
                    time.sleep(0.2)
                else:
                    if auxN != int(Datos[1]):
                        auxN = int(Datos[1])

                    for i in range(auxN):
                        x.append(int(xI2CTx.get()))
                        y.append(int(yI2CTx.get()))
                        z.append(int(zI2CTx.get()))

                    PromedioX = statistics.mean(x); PromedioY = statistics.mean(y); PromedioZ = statistics.mean(z)

                    if ser:
                        Cadena = 'Promedio X = ' + str(PromedioX) + 'Promedio Y = ' + str(PromedioY) + 'Promedio Z = ' + str(PromedioZ)
                        print(Cadena)
                        ser.write(Cadena)
                    else:
                        print('PROMEDIO INVALIDO')

```

Figura 14: Definición de función de transmisión de datos.

Se inicializa: la cadena serial, definiendo el primer valor de N con 10 muestras para tomar el promedio, igualmente se inician las listas x, y y z cuya función es la de albergar los datos de aceleración cada vez que sean tomados de las colas LIFO. Una vez se inician los threads en la función mainThread(), al iterar sobre el while(1) se inicializan las listas ya mencionadas y se revisa si la cola LIFO RxTx, la cual recibe la trama de datos, si esta se encuentra vacía, se duerme la función y se pasa a otro thread, si está llena, se toman sus datos en la variable N y se realiza un split para poder, capturar el valor de muestras (N), visualizarlos y manipularlos después. Una vez se tenga la cadena tomada de RxTx, se revisa si la trama recibida es correcta en el condicional, si esta es válida, se revisa si hay datos en la cola LIFO de cualquiera de los ejes de aceleración, en el caso de que hayan datos, se revisa que la cantidad N de muestras a tomar en el I2C sea diferente o igual, para así redefinir su valor y hacer que el sistema funcione de forma atómica respecto a esta variable de cantidad de toma de datos N. Seguido se anexan los datos de los LIFOS de aceleración a las listas para poder operar con statistics.mean() para calcular el promedio de cada eje de forma automática. Finalmente se imprimen en pantalla los promedios de aceleración y se envían por Tx al PC.

Para evitar que el valor N no sea atómico, se utiliza una variable auxiliar (auxN) para ejecutar el ciclo donde se capturan los datos para calcular el promedio, con esto garantizamos que si llega otro valor N mientras que se está ejecutando el ciclo, este no se vea afectado, y tome este nuevo valor N en la siguiente iteración.

5. Pruebas

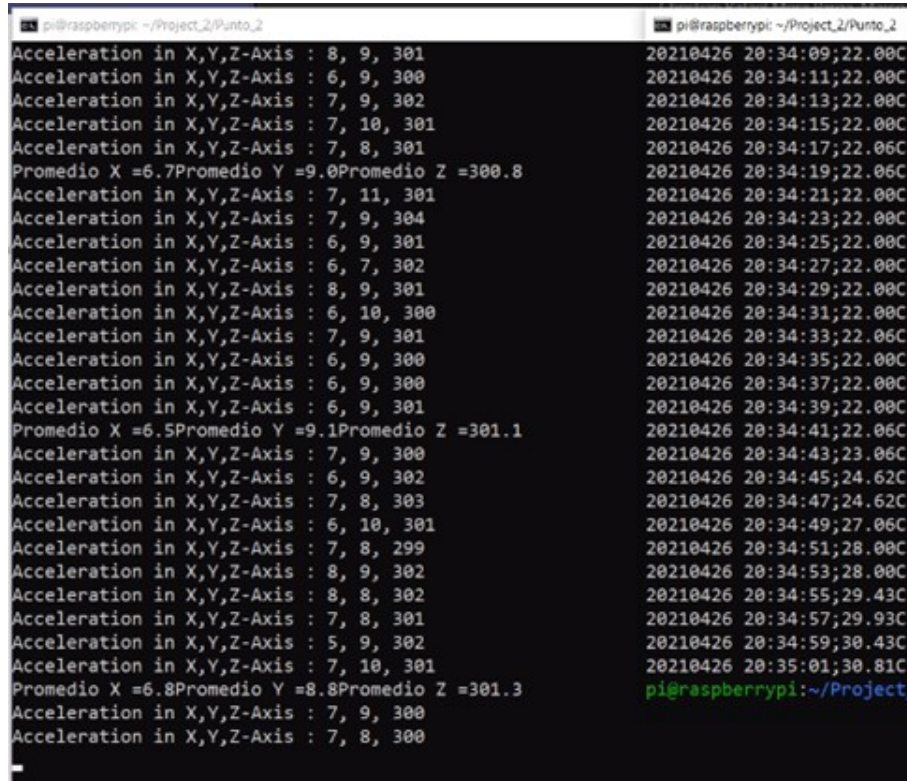
Para verificar el correcto funcionamiento del programa se ejecutan las siguientes pruebas:

Para el objetivo uno, se realiza el cálculo del tiempo que debe permanecer en un nivel alto y bajo la señal de conmutación del puerto, posterior a esto se baja la frecuencia a una que permita ser visible con el encendido y apagado de un led para garantizar la conmutación de estados (UP y DOWN), ya que no fue posible medir la respuesta de la frecuencia utilizando osciloscopio, se dejan los cálculos teóricos utilizados.

Para el desarrollo del segundo objetivo se ejecutan diferentes pruebas: primero se verificaron de forma independiente el funcionamiento de las funciones del serial (Rx y Tx), junto con las capturas de datos de aceleración (I2C), temperatura (1-Wire). Para probar la comunicación serial se verificó que el puerto estuviera funcionando correctamente enviando y recibiendo data, haciéndolo desde un sistema externo (PC) con una ventana serial de la aplicación real term. Se verificó que el promedio estuviese bien calculado realizando el cálculo a mano. Se verificó que la cadena llegase bien, enviando tramas incorrectas a la Raspberry observando que el sistema las omitiera. Para verificar el funcionamiento de las colas, se imprimieron el contenido de las posiciones de cada cola. Se hizo seguimiento de los hilos mediante las salidas stdout de python en cada etapa de inicio y final de las funciones albergadas por cada hilo. Se verificó la atomicidad de las N datos solicitados, enviando tramas con diferentes valores de N en medio de las iteraciones de captura de datos del sensor I2C observando se tomarán los N datos imprimiendo estos.

6. Resultados y Análisis

Se obtiene la cantidad inicial de valores esperados como se demuestra en la siguiente imagen, así el programa inicia generando un promedio con $N = 10$ datos que se selecciona como valor inicial.



```
pi@raspberrypi: ~/Project_2/Punto_2
Acceleration in X,Y,Z-Axis : 8, 9, 301
Acceleration in X,Y,Z-Axis : 6, 9, 300
Acceleration in X,Y,Z-Axis : 7, 9, 302
Acceleration in X,Y,Z-Axis : 7, 10, 301
Acceleration in X,Y,Z-Axis : 7, 8, 301
Promedio X =6.7Promedio Y =9.0Promedio Z =300.8
Acceleration in X,Y,Z-Axis : 7, 11, 301
Acceleration in X,Y,Z-Axis : 7, 9, 304
Acceleration in X,Y,Z-Axis : 6, 9, 301
Acceleration in X,Y,Z-Axis : 6, 7, 302
Acceleration in X,Y,Z-Axis : 8, 9, 301
Acceleration in X,Y,Z-Axis : 6, 10, 300
Acceleration in X,Y,Z-Axis : 7, 9, 301
Acceleration in X,Y,Z-Axis : 6, 9, 300
Acceleration in X,Y,Z-Axis : 6, 9, 300
Acceleration in X,Y,Z-Axis : 6, 9, 301
Promedio X =6.5Promedio Y =9.1Promedio Z =301.1
Acceleration in X,Y,Z-Axis : 7, 9, 300
Acceleration in X,Y,Z-Axis : 6, 9, 302
Acceleration in X,Y,Z-Axis : 7, 8, 303
Acceleration in X,Y,Z-Axis : 6, 10, 301
Acceleration in X,Y,Z-Axis : 7, 8, 299
Acceleration in X,Y,Z-Axis : 8, 9, 302
Acceleration in X,Y,Z-Axis : 8, 8, 302
Acceleration in X,Y,Z-Axis : 7, 8, 301
Acceleration in X,Y,Z-Axis : 5, 9, 302
Acceleration in X,Y,Z-Axis : 7, 10, 301
Promedio X =6.8Promedio Y =8.8Promedio Z =301.3
Acceleration in X,Y,Z-Axis : 7, 9, 300
Acceleration in X,Y,Z-Axis : 7, 8, 300
20210426 20:34:09;22.00C
20210426 20:34:11;22.00C
20210426 20:34:13;22.00C
20210426 20:34:15;22.00C
20210426 20:34:17;22.06C
20210426 20:34:19;22.06C
20210426 20:34:21;22.00C
20210426 20:34:23;22.00C
20210426 20:34:25;22.00C
20210426 20:34:27;22.00C
20210426 20:34:29;22.00C
20210426 20:34:31;22.00C
20210426 20:34:33;22.06C
20210426 20:34:35;22.00C
20210426 20:34:37;22.00C
20210426 20:34:39;22.00C
20210426 20:34:41;22.06C
20210426 20:34:43;23.06C
20210426 20:34:45;24.62C
20210426 20:34:47;24.62C
20210426 20:34:49;27.06C
20210426 20:34:51;28.00C
20210426 20:34:53;28.00C
20210426 20:34:55;29.43C
20210426 20:34:57;29.93C
20210426 20:34:59;30.43C
20210426 20:35:01;30.81C
pi@raspberrypi:~/Project_2
```

Figura 15: Prueba del puerto serial y funcionamiento de threads.

De la figura 15 también se observa el funcionamiento del hilo de temperatura, con lo cual se evidencia el funcionamiento de los hilos en el programa.

Como se menciona en las pruebas, una vez el programa termina la lectura de los 10 datos, procede a realizar el calculo del promedio de las lecturas y lo envía automáticamente a través del puerto serial al computador como se aprecia en la siguiente figura:

```
pi@raspberrypi:~/Project_2/Punto_2 $ pi@raspberrypi:~/Project_2/Punto_2 $ python Punto_2.py
Acceleration in X,Y,Z-Axis : 11, 7, 301
Acceleration in X,Y,Z-Axis : 11, 8, 301
Acceleration in X,Y,Z-Axis : 11, 7, 300
Acceleration in X,Y,Z-Axis : 10, 7, 300
Acceleration in X,Y,Z-Axis : 12, 7, 301
Acceleration in X,Y,Z-Axis : 11, 7, 301
Acceleration in X,Y,Z-Axis : 11, 7, 301
Acceleration in X,Y,Z-Axis : 12, 7, 302
Acceleration in X,Y,Z-Axis : 11, 7, 300
Acceleration in X,Y,Z-Axis : 12, 7, 302
Promedio X =11.2Promedio Y =7.1Promedio Z =300.9
```

Figura 16: Prueba del puerto serial.

Si se desea cambiar el número de ventana de datos tomados para el calculo del valor promedio (N), se envía la trama de `##PROMEDIO-NNN-##/n`, hacia la Raspberry. En la siguiente imagen se muestra una prueba con la trama enviada de `NNN = 006`, una vez recibida la trama, el sistema espera a finalizar el ciclo que esta en ejecución y procede a realizar el calculo del promedio con el nuevo N en la siguiente iteración o ciclo de ejecución.

```
Acceleration in X,Y,Z-Axis : 6, 8, 300
Acceleration in X,Y,Z-Axis : 6, 8, 301
Acceleration in X,Y,Z-Axis : 6, 7, 301
Promedio X =6.0Promedio Y =7.66666666667Promedio Z =300.666666667
Acceleration in X,Y,Z-Axis : 7, 7, 300
###PROMEDIO-006-##\n
Acceleration in X,Y,Z-Axis : 6, 7, 303
Acceleration in X,Y,Z-Axis : 8, 7, 300
Promedio X =7.0Promedio Y =7.0Promedio Z =301.0
Acceleration in X,Y,Z-Axis : 7, 7, 302
Acceleration in X,Y,Z-Axis : 5, 8, 303
Acceleration in X,Y,Z-Axis : 6, 7, 301
Acceleration in X,Y,Z-Axis : 6, 7, 301
Acceleration in X,Y,Z-Axis : 6, 7, 300
Acceleration in X,Y,Z-Axis : 5, 8, 301
Promedio X =5.83333333333Promedio Y =7.33333333333Promedio Z =301.333333333
Acceleration in X,Y,Z-Axis : 5, 7, 302
Acceleration in X,Y,Z-Axis : 6, 8, 301
Acceleration in X,Y,Z-Axis : 6, 7, 302
Acceleration in X,Y,Z-Axis : 6, 8, 300
Acceleration in X,Y,Z-Axis : 6, 7, 301
Acceleration in X,Y,Z-Axis : 6, 8, 300
###PROMEDIO-009-##\n
Promedio X =5.83333333333Promedio Y =7.5Promedio Z =301.0
Acceleration in X,Y,Z-Axis : 7, 7, 301
Acceleration in X,Y,Z-Axis : 6, 7, 300
Acceleration in X,Y,Z-Axis : 6, 7, 303
Acceleration in X,Y,Z-Axis : 6, 8, 302
Acceleration in X,Y,Z-Axis : 5, 8, 301
Acceleration in X,Y,Z-Axis : 6, 7, 301
Acceleration in X,Y,Z-Axis : 7, 7, 302
Acceleration in X,Y,Z-Axis : 6, 7, 300
Acceleration in X,Y,Z-Axis : 6, 6, 301
Promedio X =6.11111111111Promedio Y =7.11111111111Promedio Z =301.222222222
```

Figura 17: Prueba del puerto serial.

El mismo proceso anterior se realiza con $'NNN' = 009$, como se puede observar en la figura 17, la ejecución del programa es correcto evitando atomicidad.

Mediante el uso de hilos en el proyecto, se logra el aprovechamiento completo del procesador del sistema, ejecutando diferentes funciones durante los puntos de pausa de funciones en espera de información.

7. Conclusiones

- La ejecución de dos procesos en segundo plano, genera que el sistema operativo realice el proceso de manejo de recursos para el procesador, el cual da la apariencia de una ejecución en simultaneo (paralelo) de ambas tareas.
- El uso de hilos permite optimizar los recursos del procesamiento evitando al máximo los tiempos de espera en la ejecución del programa. Esto anterior se puede evidenciar en la activación de los programas de conmutación de los puertos GPIO en el primer punto, sometiendo ambas ejecuciones a tiempos de espera de 2.5 micro segundos, en donde si el primero se encuentra detenido, el procesador continúa ejecutando la tarea del segundo programa en otro hilo. Así mismo, en el segundo punto, se evidencia con más claridad el potencial de su uso en términos de optimización de tiempos de procesamiento, al permitir que el programa discrimine a través de los hilos, qué operaciones ejecutar, esperando así, que se completen o se cumplan condiciones e ir saltando el procesamiento entre hilos.
- Para el uso de colas e hilos es necesario ser cuidadoso con las zonas (recursos, variables, listas, métodos, etc) de atomicidad, para no corromper el estado de los datos que puedan ser usados por dos o más hilos.

8. Anexos

El código utilizado para el desarrollo del proyecto se encuentra en el siguiente link:

<https://github.com/ChristianMora-art/Sistemas-Embebidos/tree/main/Proyecto2>