

# Proyecto 1

## Sistemas Embebidos

Daniel Santiago Bonilla Betancourth  
e-mail: [daniel-bonillab@javeriana.edu.co](mailto:daniel-bonillab@javeriana.edu.co)

09 de Septiembre

### Resumen

En este informe se recopilan los resultados obtenidos en el ejercicio de práctica del Proyecto 1 de la asignatura de Sistemas Embebidos. Cómo un lenguaje de programación afecta en los resultados y recursos o cómo adquirir datos desde una Raspberry, son temas que se abordan en el trabajo.

### Introducción

Una computadora de una sola placa o *Single Board Computer* (SBC), rescata las características más importantes de una computadora convencional de escritorio desarrollada con múltiples tarjetas. Las SBC vienen incorporadas con un microprocesador, memoria, una serie de funcionalidades que no dependen de expansiones o periféricos extras y pines de entrada/salida, más conocidos como *General Purpose Input Output* (GPIO). La SBC Raspberry Pi 4 cuenta con 40 GPIO, en donde algunos de ellos tienen un propósito específico y otros son de propósito general, permitiendo al programador adaptarlos a sus necesidades, como para la lectura de sensores o para controlar algún actuador.

En la primera parte de este informe, se emplean algunos códigos en los lenguajes de programación *c++*, *bash* y *python3* para activar y desactivar un pin de la Raspberry como salida, permitiendo comprender como se ve afectado un proceso según el lenguaje, además del consumo de memoria para ejecutarla. En la segunda parte, se emplea un sensor de temperatura que emplea el protocolo de comunicación 1-Wire, logrando que el estudiante adquiera los fundamentos para el manejo de GPIO en una SBC, además de una adecuada adquisición y almacenamiento de datos.

### Arquitectura de Bloques General (ABG)

#### ABG punto 1

La configuración de los componentes para el primer punto se puede visualizar en la Figura 1. Desde la raspberry se corren los códigos escritos en *c++*, *bash* y *python3* que alternan el estado de salida del GPIO 23. Según la nomenclatura con la que se trabaje, el GPIO 23 es el pin 16 de la tarjeta, el pin 4 desde la librería Wiring Pi o el GPIO 23 desde BCM. Se lee esta señal con Osciloscopio y así conocer la frecuencia a la que comuta.



Figura 1: ABG del punto 1

## ABG punto 2

Para el punto 2, se configuran los componentes como se ve en la Figura 2. Se emplea el sensor de temperatura DS18B20, el cual se comunica con la SBC Raspberry a través del protocolo 1-Wire. Este sensor requiere de una resistencia en *pull-up* y una alimentación al pin de 3.3V para su correcto funcionamiento. Raspberry ya tiene el GPIO 4 configurado por defecto para este tipo de comunicación, el cual es empleado para la lectura de datos a través de un script realizado en *bash* que se encarga, no solo de leer sino, de almacenar los datos en un documento *.csv*.

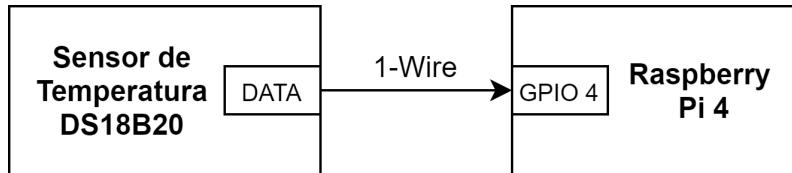


Figura 2: ABG del punto 2

## Desarrollo de la solución

### Desarrollo del punto 1

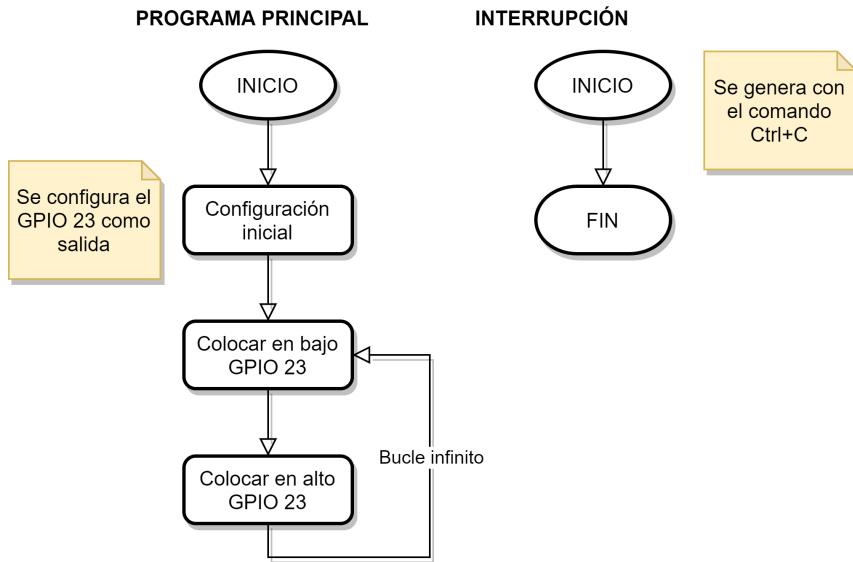
En la primera parte del proyecto, se emplea la lógica presente en el diagrama de flujo de la Figura 3. Independientemente del lenguaje, se siguen las etapas descritas en el diagrama para hacer que uno de los pines de la Raspberry commute su estado de BAJO a ALTO una y otra vez. Este proceso solo se detendrá al matar el proceso desde la terminal.

Algunas consideraciones según el lenguaje de programación:

- En *c++* se emplean las librerías `wiringPi.h` y `iostream`. Luego de realizar la configuración inicial de `wiringPi` con el comando `wiringPiSetup()`, se debe recordar que esta librería emplea una nomenclatura para los pines de la Raspberry, en este caso el GPIO 23 corresponde al pin 4 y es justo este el que se declara dentro de la función `pinMode` como salida. Luego se entra a un bucle infinito para iterar los estados de este pin a través de un `digitalWrite`.
- En *bash* se realiza primero un `unexport` del GPIO 23, en caso de estar siendo usado o no haberse cerrado bien alguna ejecución anterior con ese pin. Luego de eso, si se exporta el pin. Los comandos para estas dos tareas son `echo 23 > /sys/class/gpio/unexport` y `echo 23 > /sys/class/gpio/export`. Una vez llamado el pin de interés, se debe declarar como salida con el comando `echo out > /sys/class/gpio/gpio23/direction`. Finalmente, se entra en un bucle infinito que le asigna el valor de alto o bajo al pin con los números 1 y 0 respectivamente a través del comando `echo <1 ó 0> > /sys/class/gpio/gpio23/value`
- En *python3* se trabaja con la librería `RPi.GPIO`. Se puede declarar el pin de trabajo desde la nomenclatura de GPIO, pero para revisar el escenario de pines físicos, se declara el pin a usar como el pin 16 físico, que corresponde al GPIO 23 usado anteriormente. Esto se hace con el comando `GPIO.setmode(GPIO.BOARD)` seguido de `GPIO.setup(PIN,GPIO.OUT)` que lo coloca como salida. Con el comando `GPIO.output(PIN, <VALUE>)`, se puede modificar el valor de salida del pin, colocando un 1 o 0 en el parámetro de VALUE. Esto se realiza dentro de un bucle infinito.

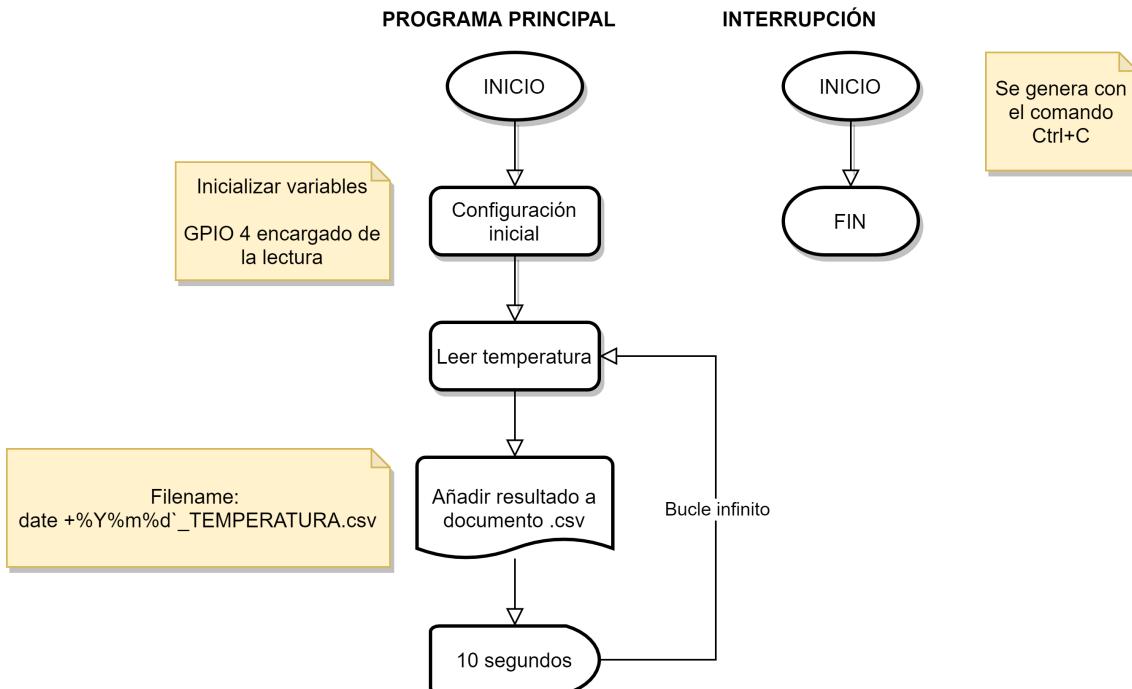
### Desarrollo del punto 2

El segundo punto también presenta un diagrama de flujo sencillo. En la Figura 4 se contempla la lógica aplicada en el código realizado. Se genera una variable de apoyo para conocer desde la terminal en que iteración va el código escrito en *bash*. Para comenzar con la toma de datos de temperatura se asigna en una variable el valor registrado en la dirección `/sys/bus/w1/devices/28-3c01e076c7cc/temperature`, el cual registra el último valor detectado por el sensor. Esta temperatura se registrará en un archivo denominado `YYYYMMDD_TEMPERATURA.csv`, en



**Figura 3:** Diagrama de flujo del punto 1

donde YYYY, MM y DD corresponden al año, mes y día respectivamente de cuando se registraron los datos. Además, dentro de ese documento, se contará con dos columnas en donde la primera corresponde a la fecha más la hora, minuto y segundo de cuando se tomó la temperatura, pues recordemos que se registrará la temperatura cada 10 segundos, y la segunda tendrá la temperatura en miligrados Celsius.



**Figura 4:** Diagrama de flujo del punto 2

# Pruebas

## Protocolo de pruebas punto 1

### Elementos de trabajo

- Raspberry Pi 4
- Osciloscopio
- Sonda
- Jumpers

### Procedimiento

Teniendo la configuración descrita en la Figura 1, se conecta la sonda al osciloscopio y se mide la señal entre el GPIO configurado y la tierra de la tarjeta. Se debe ajustar la frecuencia y amplitud del Osciloscopio para poder ver como se comuta la señal de acuerdo al código empleado. Se puede frenar el proceso con el comando **Ctrl+C**

### Resultados esperados

Ver una señal que vislumbren la comutación de estado del GPIO empleado. Dicha señal debe aproximarse a una señal periódica.

## Protocolo de pruebas punto 2

### Elementos de trabajo

- Raspberry Pi 4
- Sensor DS18B20
- Resistencia 4.7 kOhms
- Jumpers

### Procedimiento

Con la configuración descrita en la Figura 2, se ejecuta el script de este punto mientras que se varía la temperatura del sensor. Esto se debe realizar durante varios minutos, en la terminal aparecerá cuantas mediciones se llevan tomadas desde que arranco el script a correr. Se puede frenar el proceso con el comando **Ctrl+C**

### Resultados esperados

Un archivo .csv donde se encuentren registradas cada una de las temperaturas, con un delay de 10 segundos entre muestras.

# Resultados y Análisis

## Resultados punto 1

Lenguaje	Librería	Tiempo de comutación	Frecuencia
C++	wiringPi	33.85 ns	29.54 MHz
Bash	-	107.8 $\mu$ s	9.277 KHz
Python3	RPi.GPIO	1.220 $\mu$ s	819.7 KHz

Tabla 1: Resultados del punto 1 para cada lenguaje

De la Tabla 1 se deduce que el código escrito en *C++* es el más rápido de todos. Realiza la conmutación 3184 veces más rápido que en *bash* y 36 veces más rápido que en *python*. En cuanto a la comparación entre los otros dos lenguajes, resulta que *python* fue 88 veces más rápido que en *bash*.

En cuanto al uso de memoria, los 3 códigos consumen el 100% de la CPU, es decir, emplea un procesador completamente para llevar a cabo dicha tarea. Además, la Raspberry a veces alterna el procesador encargado de la tarea.

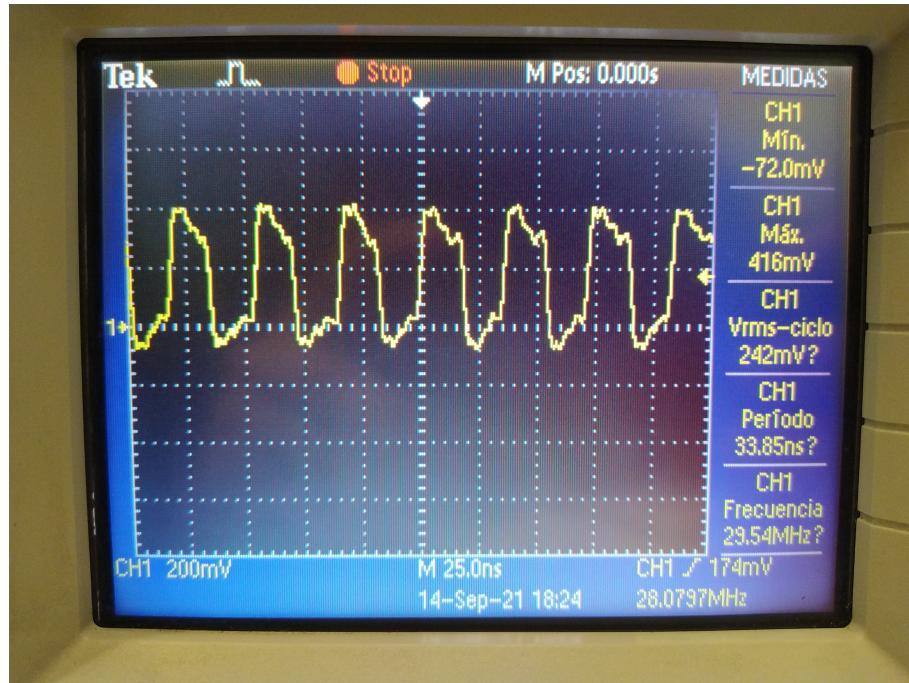


Figura 5: Señal de salida por el código en *C++*

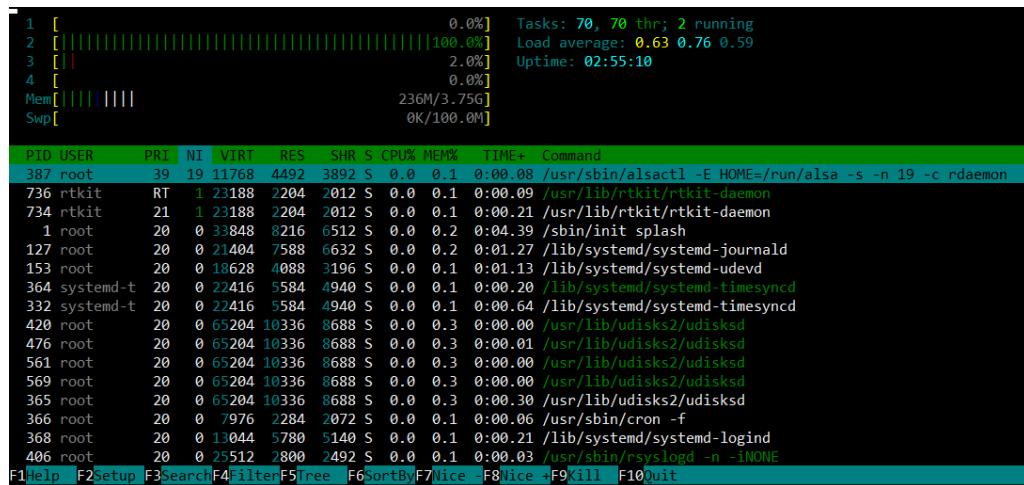


Figura 6: Memoria empleada por el código en *C++*

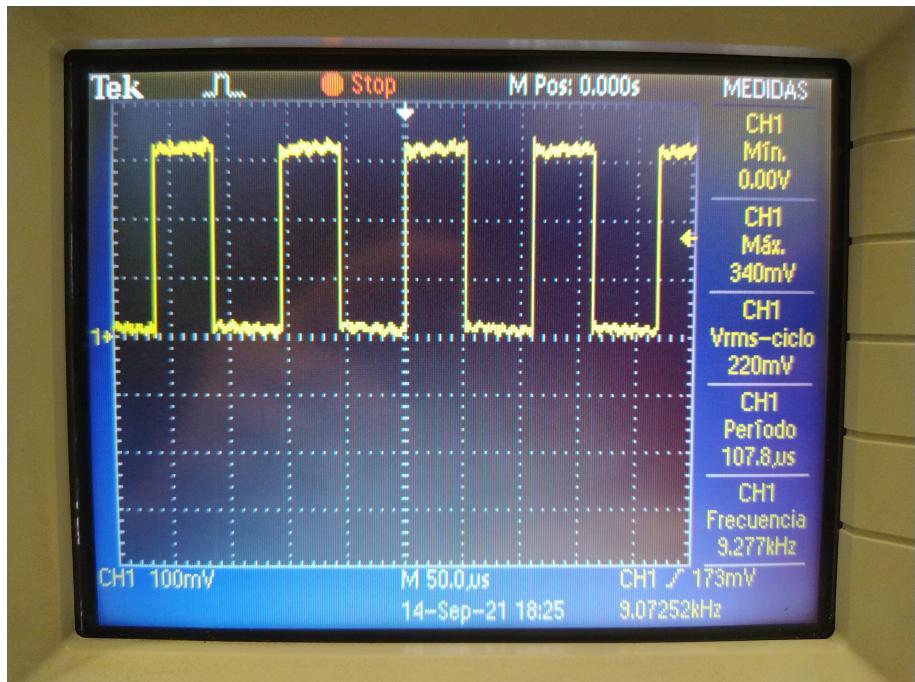


Figura 7: Señal de salida por el código en *bash*

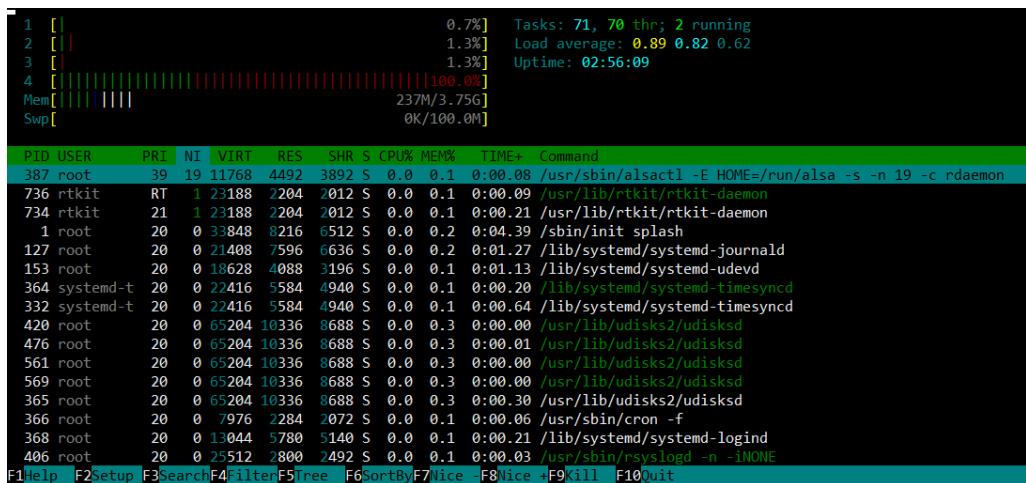
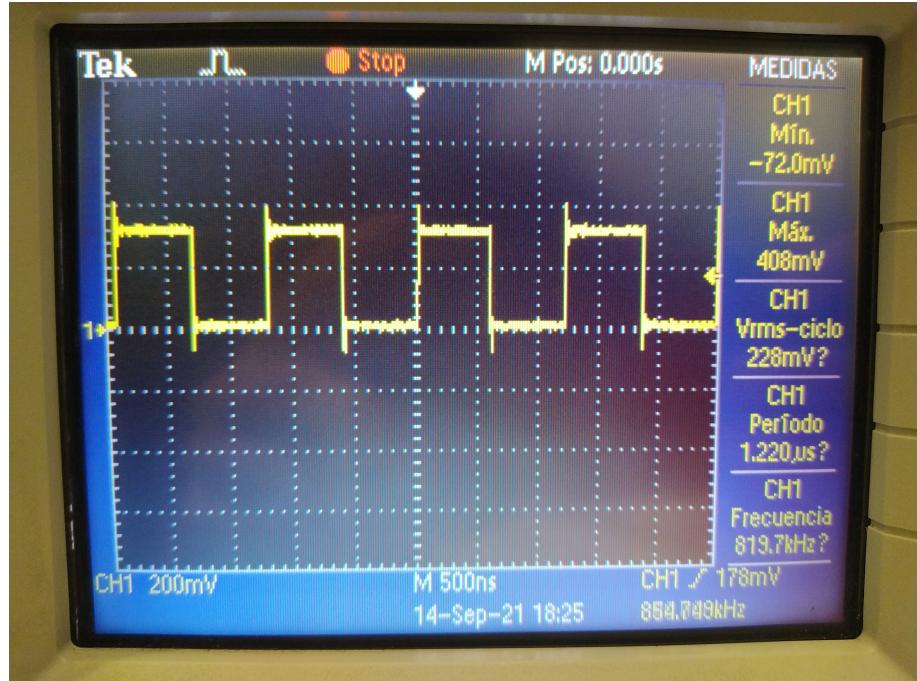


Figura 8: Memoria empleada por el código en *bash*



**Figura 9:** Señal de salida por el código en *python3*

```

1 [          0.0%] Tasks: 70, 70 thr; 2 running
2 [|||        1.3%] Load average: 0.99 0.86 0.65
3 [|||        1.3%] Uptime: 02:57:04
4 [|||||     100.0%]
Mem[||||| 238M/3.75G]
Swp[          0K/100.0M]

PID USER    PRI  NI   VIRT   RES   SHR S CPU% MEM% TIME+  Command
387 root      39  19 11768 4492 3892 S  0.0  0.1  0:00.08 /usr/sbin/alsactl -E HOME=/run/alsa -s -n 19 -c rdaemon
736 rtkit     RT   1 23188 2204 2012 S  0.0  0.1  0:00.09 /usr/lib/rtkit/rtkit-daemon
734 rtkit     21   1 23188 2204 2012 S  0.0  0.1  0:00.21 /usr/lib/rtkit/rtkit-daemon
  1 root      20   0 33848 8216 6512 S  0.0  0.2  0:04.39 /sbin/init splash
127 root      20   0 21408 7600 6640 S  0.0  0.2  0:01.28 /lib/systemd/systemd-journald
153 root      20   0 18628 4088 3196 S  0.0  0.1  0:01.13 /lib/systemd/systemd-udevd
364 systemd-t 20   0 22416 5584 4940 S  0.0  0.1  0:00.20 /lib/systemd/systemd-timesyncd
332 systemd-t 20   0 22416 5584 4940 S  0.0  0.1  0:00.64 /lib/systemd/systemd-timesyncd
420 root      20   0 65204 10336 8688 S  0.0  0.3  0:00.00 /usr/lib/udisks2/udisksd
476 root      20   0 65204 10336 8688 S  0.0  0.3  0:00.01 /usr/lib/udisks2/udisksd
561 root      20   0 65204 10336 8688 S  0.0  0.3  0:00.00 /usr/lib/udisks2/udisksd
569 root      20   0 65204 10336 8688 S  0.0  0.3  0:00.00 /usr/lib/udisks2/udisksd
365 root      20   0 65204 10336 8688 S  0.0  0.3  0:00.30 /usr/lib/udisks2/udisksd
366 root      20   0 7976 2284 2072 S  0.0  0.1  0:00.06 /usr/sbin/cron -f
368 root      20   0 13044 5780 5140 S  0.0  0.1  0:00.21 /lib/systemd/systemd-logind
406 root      20   0 25512 2800 2492 S  0.0  0.1  0:00.03 /usr/sbin/syslogd -n -iNONE
F1?help F2Setup F3Search F4FilterTree F6SortBy F7Nice F8!Nice +F9!Kill F10Quit

```

**Figura 10:** Memoria empleada por el código en *python3*

## Resultados punto 2

```
pi@raspberrypi:~/Documents/Proyecto1/Punto2 $ cat 20210915_TEMPERATURA.csv
20210915 180629;25250
20210915 180640;26062
20210915 180651;27812
20210915 180702;28437
20210915 180713;28750
20210915 180724;28937
20210915 180735;29062
20210915 180746;29187
20210915 180757;29250
20210915 180808;29312
20210915 180819;29062
20210915 180830;28687
20210915 180841;28250
20210915 180852;28000
20210915 180902;27625
20210915 180913;27250
20210915 180924;26937
20210915 180935;26625
20210915 180946;26312
20210915 180957;26000
20210915 181008;25750
20210915 181019;25500
20210915 181030;25250
20210915 181041;25062
20210915 181052;24812
20210915 181103;24625
20210915 181114;26875
```

Figura 11: Resultado del punto 2. Registro de temperaturas

En la Figura 11 se ve el registro obtenido de temperatura con el sensor. En la primera columna aparece la fecha con la hora y seguido del punto y coma tenemos la temperatura en miligrados Celsius. Si miramos con detenimiento la primera columna, y nos centramos en los dos últimos dígitos que corresponden a los segundos, se ve que hay un registro cada 11 segundos, y no 10 segundos como fue programado.

## Conclusiones

- Al manipular los pines de una SBC se debe tener cuidado con la nomenclatura con la que se trabaja, pues puede suceder que un pin tenga diferentes números para dos nomenclaturas diferentes.
- La programación en lenguajes de bajo nivel permite obtener una ejecución más rápida, empleando mejor los recursos del sistema. Por el contrario, en los lenguajes de alto nivel se pierde velocidad pero se gana facilidad en su programación. Además, los lenguajes de más alto nivel suelen poseer una comunidad más grande que contribuye con librerías para el fácil manejo de periféricos u otros componentes (incluso de recursos del sistema).
- La manipulación de un sensor 1-Wire resulta muy sencillo al traer todo en un solo canal. Es muy similar al protocolo de comunicación I2C, solo que en este caso 1-Wire emplea velocidades de transferencia más bajas pero con un rango más amplio.
- En el punto 2 se esperaba tener un registro cada 10 segundos, y al final resultó de 11 segundos. Para solucionar esto, se propone emplear un delay de 9 segundos dentro del script.

# Bibliografía

- [1] Raspberry Pi Pinout, W1-GPIO - One-Wire Interface, accedido en Agosto 2021, <[https://pinout.xyz/pinout/1\\_wire](https://pinout.xyz/pinout/1_wire)>
- [2] Programo Ergo Sum, Primeros pasos con Pines GPIO en Raspberry Pi, accedido en Septiembre 2021, <<https://www.programoergosum.es/tutoriales/introduccion-a-pines-gpio-en-raspbian/>>

## Herramientas:

- Diagramas de flujo: <https://www.draw.io/>
- Tablas latex: <https://www.tablesgenerator.com/>

# Anexos

## Código Cpp punto 1

```
1 #include <iostream>
2 #include <wiringPi.h>
3
4 using namespace std;
5
6 const int pin = 4;
7 int main()
8 {
9     wiringPiSetup();
10    pinMode(pin, OUTPUT);
11
12    while(1) {
13        digitalWrite(pin, 0);
14        digitalWrite(pin, 1);
15    }
16    return 0;
17 }
```

## Código Bash punto 1

```
1#!/bin/bash
2echo 23 > /sys/class/gpio/unexport
3echo 23 > /sys/class/gpio/export
4echo out > /sys/class/gpio/gpio23/direction
5
6while true
7do
8    echo 1 > /sys/class/gpio/gpio23/value
9    #sleep 5
10   echo 0 > /sys/class/gpio/gpio23/value
11   #sleep 5
12done
```

## Código Python punto 1

```
1 #!/usr/bin/env python3
2
3 import RPi.GPIO as GPIO
4 import time
5
6 #GPIO initialization
7 GPIO.setmode(GPIO.BOARD)
8 GPIO.setwarnings(False)
9
10 PIN = 16
11 #Initialize your pin
12 GPIO.setup(PIN,GPIO.OUT)
13
14 while True:
15     GPIO.output(PIN, 1)
16     #time.sleep(5)
17     GPIO.output(PIN, 0)
18     #time.sleep(5)
```

## Código punto 2 - Adquisición de temperatura

```
1#!/bin/bash
2# Autor: Santiago Bonilla
3# Sistemas Embbebidos - Proyecto 01
4
5var=0
6while true
7do
8    temperature=`cat /sys/bus/w1/devices/28-3c01e076c7cc/temperature` 
9    echo `date +%Y%m%d\ %H%M%S`';'$temperature >> ...
10   /home/pi/Documents/Proyecto01/Punto2/`date +%Y%m%d`_TEMPERATURA.csv
11   var=`echo "$var+1" | bc -l`
12   echo "Iteracion numero: " $var
13   sleep 10
14done
```