ESTGOH
Escola Superior de Tecnologia e Gestão de Oliveira do Hospital
Instituto Politécnico de Coimbra

Licenciatura em Engenharia Informática

Tecnologia e Arquitetura de Computadores 2022/2023

Trabalho Prático nº 7

Timer Library

Realizado em: 25/05/2023
Elaborado em: 26/05/2023

Grupo: 5

António Dinis - a2021157297
Francisco Figueiras - a2021155919
Mariana Magalhães - a2022147454

Índice

1. Introdução

Este trabalho tem como objetivo fazer 3 exercícios usando a função Timer1() e para podermos utilizar essa função temos de instalar a biblioteca TimerOne.

A função Timer1 refere-se a uma funcionalidade de temporização disponível em muitos microcontroladores e microprocessadores. Esta permite criar interrupções ou eventos periódicos com base num contador de tempo interno.

O Timer1 geralmente é utilizado para realizar tarefas relacionadas ao controle de tempo, como medição de intervalos, atrasos, gerar pulsos de clock, entre outros. Possui um registo de contagem que é incrementado a cada ciclo de clock do sistema e pode ser configurado para gerar uma interrupção quando atinge um valor específico.

2. Métodos

O trabalho foi realizado no decorrer das 3 horas de aula de **Tecnologia e Arquitetura de Computadores (TAC)** Para a realização deste trabalho foi utilizado o **Tinkercad**, um programa de modelagem tridimensional (3D) online e gratuito que é executado num navegador da web, conhecido por ser simples e fácil de utilizar, sendo este usado para projetar o circuito. Para além do **Tinkercad**, foi usado o **Arduino IDE**, uma plataforma de prototipagem eletrónica de hardware livre e de placa única, projetada com um microcontrolador com suporte de entrada/saída embutido, uma das linguagens de programação padrão usada no programa é C/C++, neste caso essa linguagem é usada para o desenvolvimento do código. Todos este programas foram desenvolvidos num computador com um processador AMD Ryzenn 7 5800H With Radeon Graphics e também foram usados os materiais representados nas tabelas seguintes.

Nome	Quantidade	Componente
U1	1	Arduino Uno R3
D1	1	White LED
D2	1	Blue LED
R1,R2	1	560Ω Resistor

Tabela 1 - materiais do exercício 1

Nome	Quantidade	Componente
U1	1	Arduino Uno R3
U1	1	Sensor de temperatura [TMP36]

Tabela 2 - materiais do exercício 2 e 3

3. Resultados

3.1. Exercício 1

O objetivo neste exercício é piscar dois leds em frequências diferentes. Um led deve piscar a cada segundo e o outro a cada 300 milissegundos.

Começámos por fazer o projeto do circuito no Tinkercad (Figura 1) e através dessa montagem foi obtido o diagrama do circuito (figura 2), em seguida foi desenvolvido um algoritmo para o desenvolvimento do código no Arduíno e por fim a sua montagem na breadbord (figura 3).

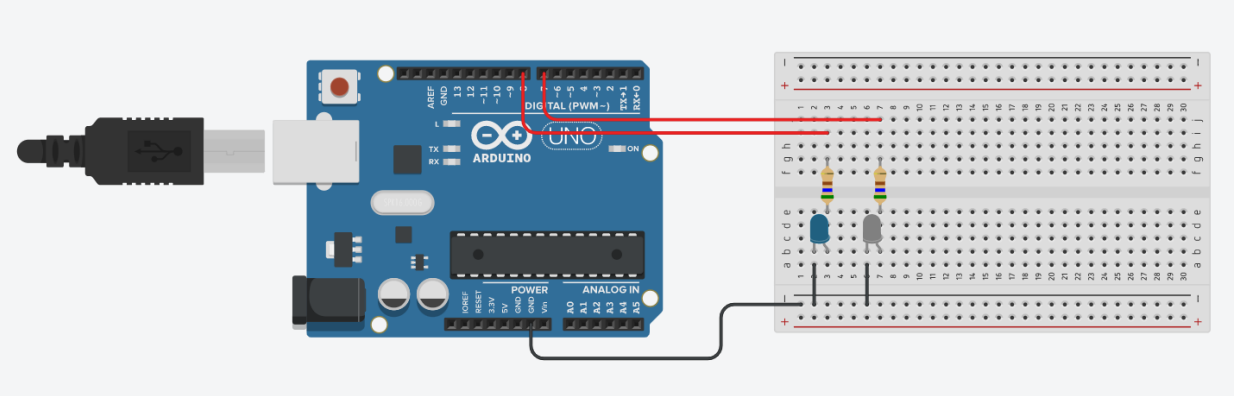

Figura 1 - montagem no tinkercad

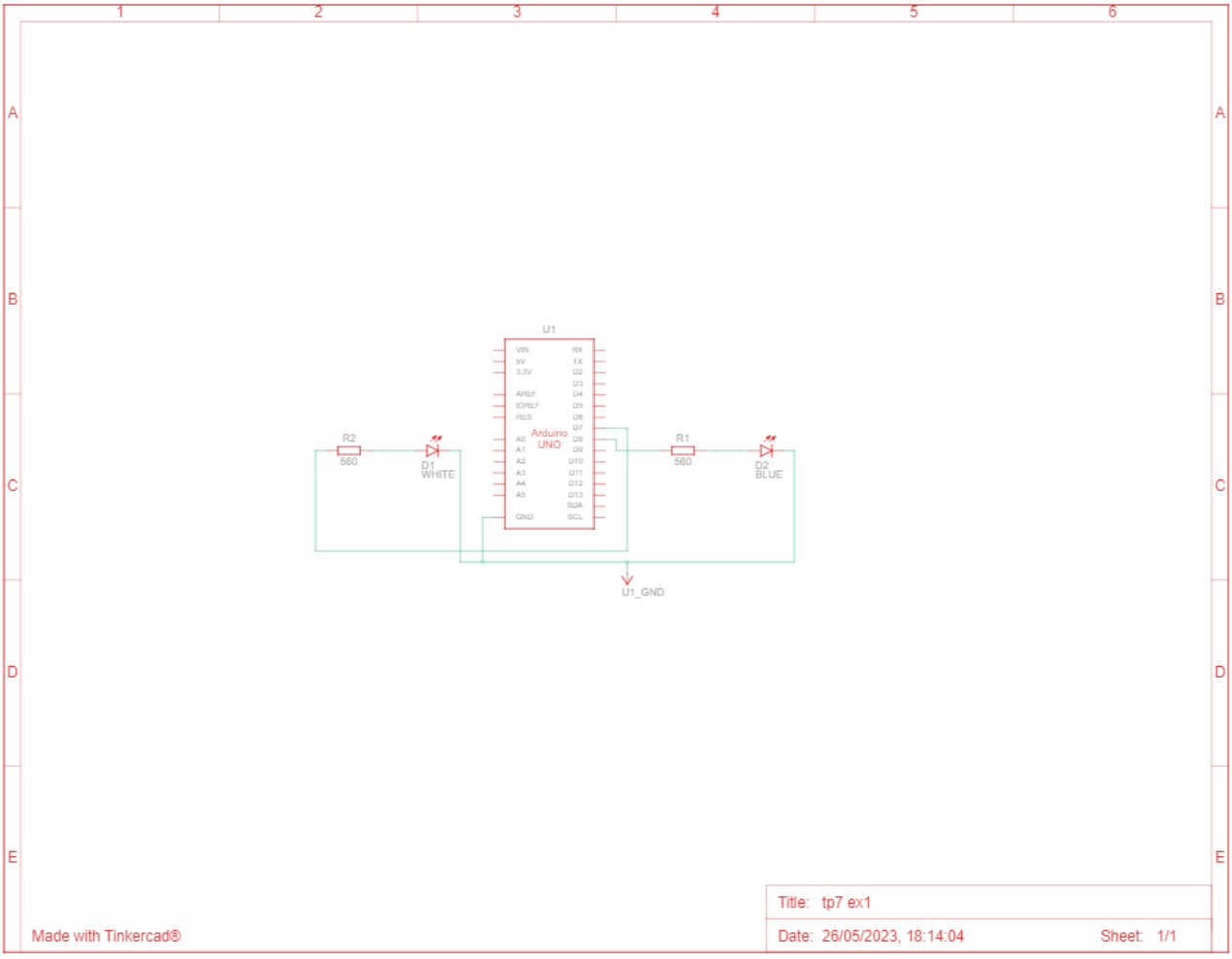

Figura 2 - diagrama do circuito

Algoritmo do código:

1. Incluir a biblioteca **TimerOne.h** para usar as funções do temporizador.
2. Definir os pinos 7 e 8 como saída usando a função **pinMode().**
3. Inicializar o temporizador **Timer1** com um tempo de 1000000 (1 segundo).
4. Anexar a interrupção **blink_timer()** ao temporizador **Timer1** usando a função **attachInterrupt()**.
5. No loop principal, chamar a função **blink_millis()** para alternar o estado do pino 8 usando a função **digitalWrite()**.
6. A função **blink_timer()** é uma interrupção que é acionada a cada segundo pelo temporizador **Timer1**. Ela inverte o estado do pino 7 usando a função **digitalWrite(`.**
7. A função **blink_millis()** verifica se passaram mais de 300 milissegundos desde a última vez que o pino 8 foi alterado. Se sim, inverte o estado do pino 8 usando **digitalWrite()** e atualiza o valor de **time** para o valor atual de **millis**().

O código representado abaixo foi o utilizado. A função **setup()** é chamada no início do programa, nesta configuramos os pinos 7 e 8 como saída usando a função **pinMode()**. De seguida, foi inicializado o temporizador **Timer1** com uma frequência de 1000000 microssegundos (1 segundo) e, por fim, anexamos a função **blink_timer()**, que faz uma interrupção que é acionada a cada segundo pelo temporizador **Timer1**, esta alterna o estado do pino 7 usando a função **digitalWrite()**. A variável **is_on** controla o estado atual do pino e é invertida a cada chamada da função. A função **loop()** é executada continuamente após o **setup()**, neste caso estamos a chamar a função **blink_millis()** dentro do loop, esta função vai verificar se passaram mais de 300 milissegundos desde a última vez que o pino 8 foi alterado, se sim, ela inverte o estado do pino 8 usando a função **digitalWrite()** e atualiza o valor de **time** para o valor atual de **millis()**. Isso garante que o pino 8 pisque a cada 300 milissegundos.

```
#include <TimerOne.h>
void blink_timer();
void blink_millis();
void setup() {
  pinMode(7, OUTPUT);
  pinMode(8, OUTPUT);
  Timer1.initialize(1000000);
  Timer1.attachInterrupt(blink_timer);
}
void loop() {
  blink_millis();
}
void blink_timer() {
  static bool is_on = false;
  digitalWrite(7, !is_on);
  is_on = !is_on;
}
void blink_millis() {
  static unsigned long time = 0;
  if (millis() - time > 300) {
    digitalWrite(8, !digitalRead(8));
    time = millis();
  }
```

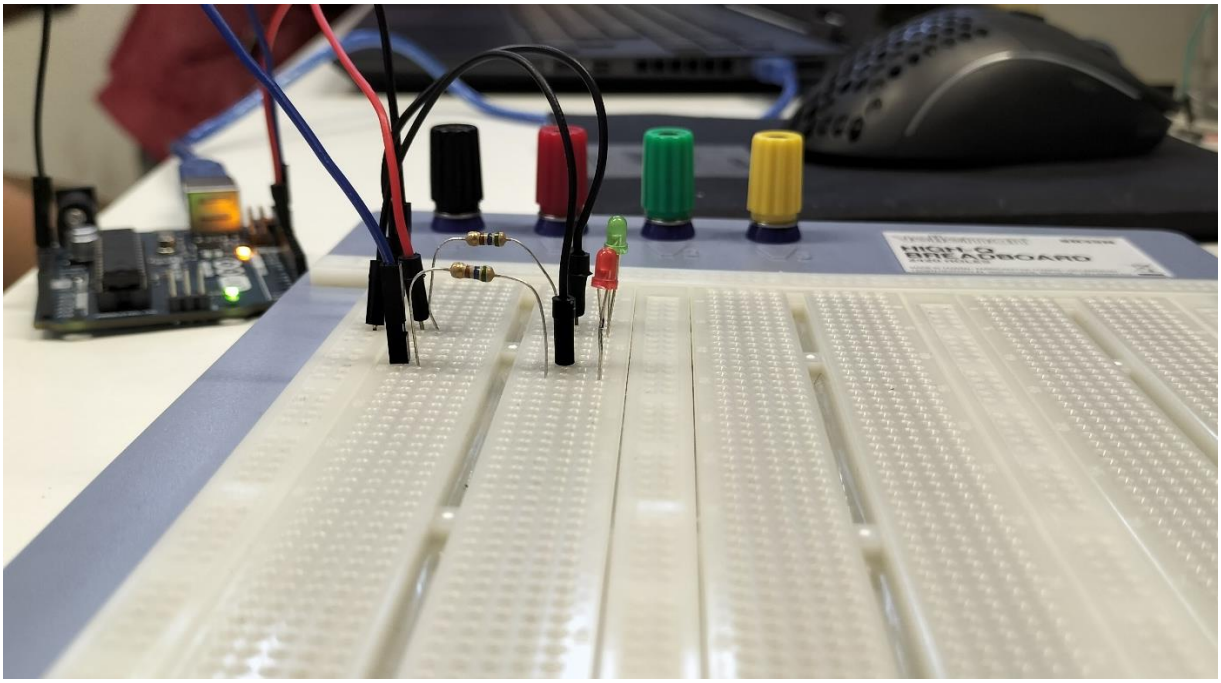

Figura 3 – bradbord

3.2. Exercício 2

O objetivo neste exercício é enviar o valor da temperatura para o PC através da porta serial a cada segundo usando a biblioteca de timer.

Começamos por fazer o projeto do circuito no Tinkercad (Figura 4) e através dessa montagem foi obtido o diagrama do circuito (figura 5), em seguida foi desenvolvido um algoritmo para o desenvolvimento do código no Arduíno e por fim a sua montagem na breadbord (figura 6).

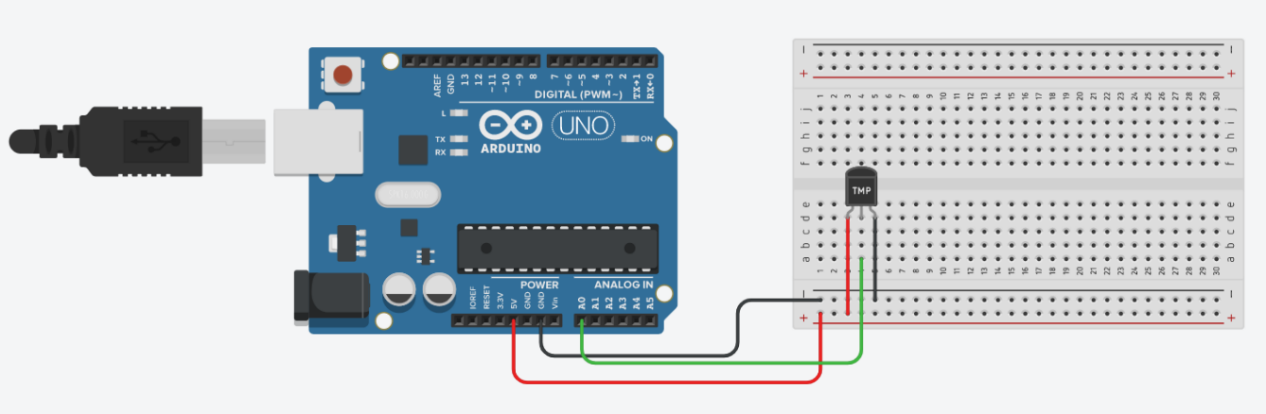

Figura 4 - montagem do sensor TMP no tinkercad

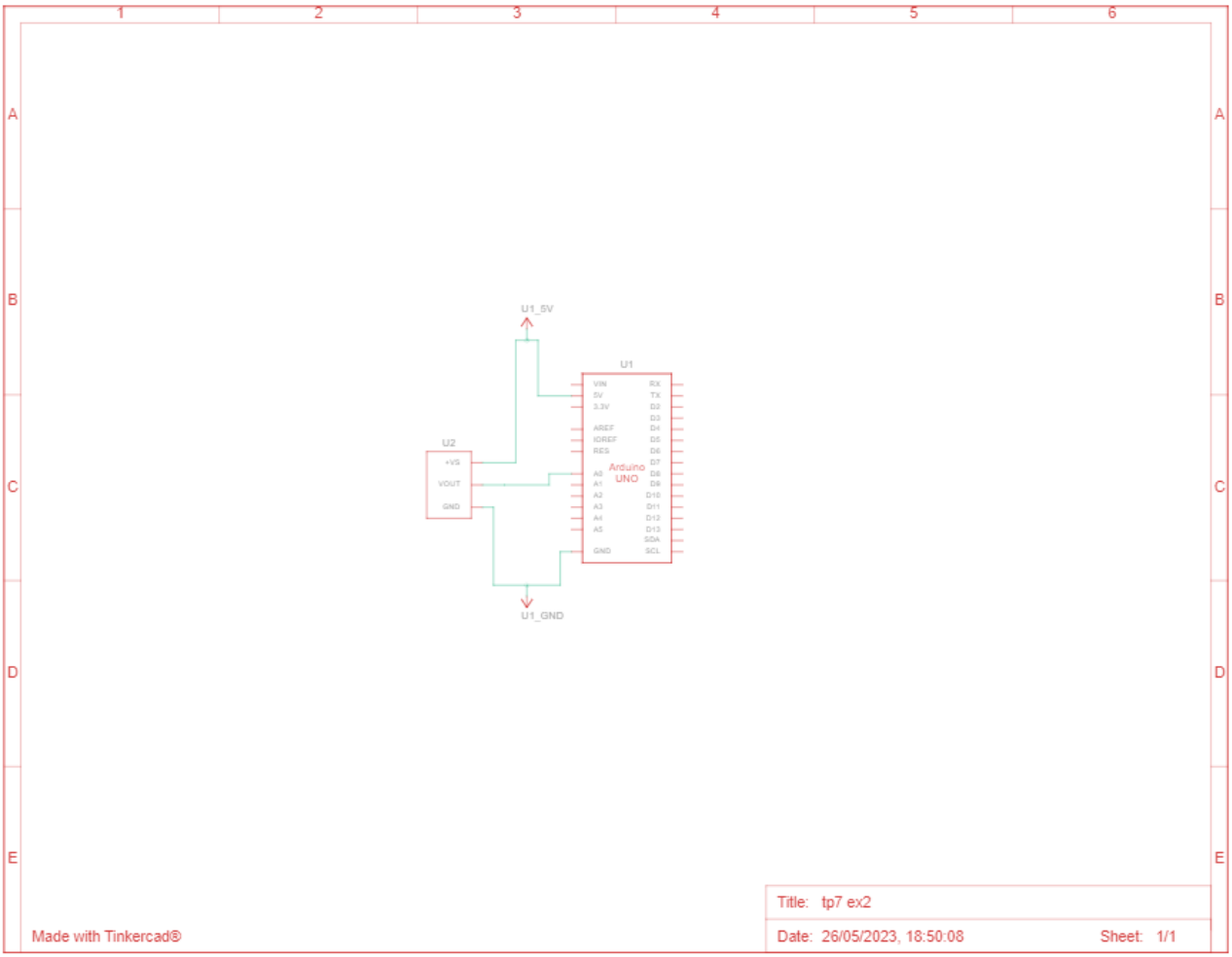

Figura 5 - diagrama do circuito

Algoritmo do circuito:

1. Incluir a biblioteca **TimerOne.h** para utilizar as funções do temporizador.
2. Definir o protótipo da função **sensor().**
3. Na função **setup()**:
   - Inicializar a comunicação serial com a taxa de transmissão de 9600 bps usando **serial.begin(9600)**.
   - Inicializar o temporizador **Timer1** com uma frequência de 1000000 microssegundos (1 segundo) usando **Timer1.initialize(1000000)**.
   - Anexar a função **sensor()** como uma interrupção ao temporizador **Timer1** usando **Timer1.attachInterrupt(sensor).**
4. Na função **loop**(), deixar vazio, pois não há nenhuma tarefa específica a ser repetida.
5. Na função **sensor()**:
   - Ler o valor analógico presente no pino A0 usando **analogRead(A0)**.
   - Converter o valor lido de uma escala de 0 a 1023 para uma escala de 0 a 5V usando a fórmula (valorLido * 5.0 / 1024.0).
   - Subtrair 0.5 do valor convertido para ajustar a faixa de -0.5V a 4.5V.
   - Multiplicar o resultado por 100 para obter a temperatura em graus Celsius.
   - Imprimir a temperatura na porta serial usando **serial.print()** para exibir o valor da temperatura e **Serial.println()** para adicionar uma quebra de linha.

O código a baixo foi o utilizado para configurar um temporizador para acionar uma interrupção a cada segundo, na qual é lido um valor analógico do pino A0 convertido em temperatura e exibido na porta serial.

```
#include <TimerOne.h>

void sensor();

void setup() {
  Serial.begin(9600);
  Timer1.initialize(1000000);
  Timer1.attachInterrupt(sensor);
}

void loop() {}

void sensor() {
  float temperature = ((analogRead(A0) * 5.0 / 1024.0) - 0.5) * 100;
  Serial.print(temperature);
  Serial.println("°C");
}
```

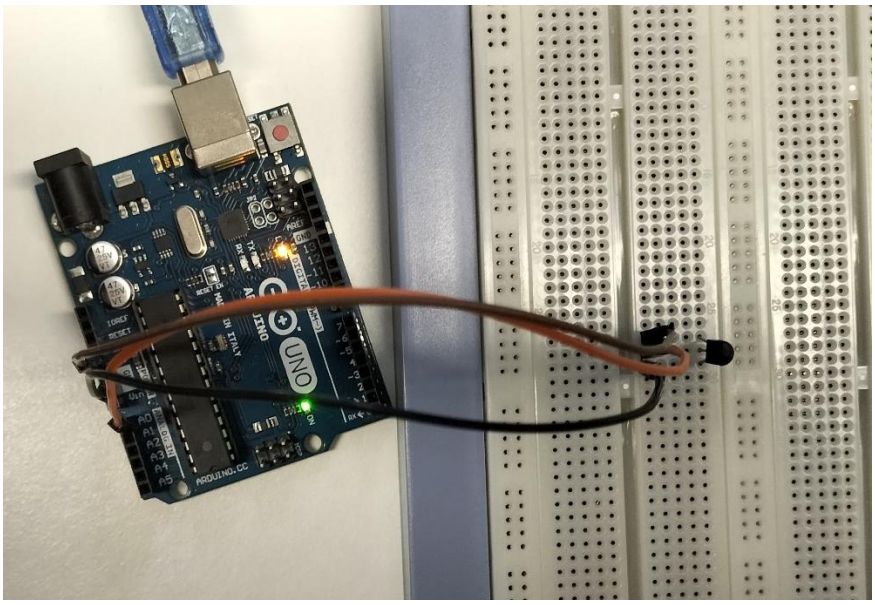

Figura 6 - breadbord

3.3. Exercício 3

O objetivo neste exercício é utilizar o exercício anterior, mas desta vez só enviar os valores mínimos, máximos, medio para o computador, guardar a data e hora e os valores recolhidos num ficheiro de texto. Este exercício foi feito em python e em C.

O código em C foi um dos mais complicados porque tivemos de desenvolver um código script para só ao fim de 1 minuto enviar os dados e guardar em texto e foi preciso instalar algumas bibliotecas como a do windows.h.

Começamos por incluir algumas bibliotecas necessárias para o programa, bem como definir algumas constantes e códigos de erro, de seguida na função **main()** as variáveis **hSerial** e **fp** são declaradas para representar o identificador do dispositivo serial e o ponteiro para o arquivo, respetivamente.

A variável **portName** é um array de caracteres usado para armazenar o nome da porta serial. A variável **errorCode** é usada para armazenar códigos de erro.

Depois pedimos ao utilizador que insira o nome porta serial, em seguida ele vai verificar se a porta serial fornecida é válida, tentando abrir a porta usando a função **CreateFile()**. Se a porta não puder ser aberta, um erro é exibido e o programa é encerrado.

Em seguida abre um arquivo para escrever os dados pretendidos. A função **openDataFile()** é chamada para criar e abrir o arquivo. Se ocorrer algum erro ao abrir o arquivo, a função retornará **NULL**, e o programa será encerrado com um código de erro não especificado.

Abaixo encontra-se o código script.

```c
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <windows.h>
#include <time.h>

#define BUFFER_SIZE 1024
// error codes
#define ERROR_OPEN_SERIAL_PORT 1
#define ERROR_CONFIGURE_SERIAL_PORT 2
#define ERROR_OPEN_FILE 3
#define ERROR_SET_COMM_TIMEOUTS 4

// prototypes
int configureSerialPort(HANDLE hSerial);
int captureData(HANDLE hSerial, FILE* fp);
FILE* openDataFile();

int main() {
    HANDLE hSerial;
    FILE* fp;
    char portName[20];
    int errorCode = 0;
```

```
    // get serial port name from user
    printf("Enter serial port name (e.g. COM3): ");
    scanf("%s", portName);
    while (getchar() != '\n'); // flush input buffer

    // verify serial port
    printf("Verifying serial port...\n");
    hSerial = CreateFile(portName, GENERIC_READ, 0, NULL, OPEN_EXISTING,
FILE_ATTRIBUTE_NORMAL, NULL);
    if (hSerial == INVALID_HANDLE_VALUE) {
        printf("Error: Failed to open serial port\n");
        getchar();
        return ERROR_OPEN_SERIAL_PORT;
    }
    CloseHandle(hSerial);

    // open data file for writing
    fp = openDataFile();
    if (fp == NULL) {
        CloseHandle(hSerial);
        getchar();
        return 1;
    }

    // open serial port
    printf("Opening serial port...\n");
    hSerial = CreateFile(portName, GENERIC_READ, 0, NULL, OPEN_EXISTING,
FILE_ATTRIBUTE_NORMAL, NULL);
    if (hSerial == INVALID_HANDLE_VALUE) {
        printf("Error: Failed to open serial port\n");
        getchar();
        return ERROR_OPEN_SERIAL_PORT;
    }

    // configure serial port
    errorCode = configureSerialPort(hSerial);
    if (errorCode != 0) {
        CloseHandle(hSerial);
        printf("Error: Failed to configure serial port\n");
        getchar();
        return errorCode;
    }

    // capture data
    errorCode = captureData(hSerial, fp);
    if (errorCode != 0) {
        fclose(fp);
        CloseHandle(hSerial);
        printf("Error: Failed to capture data\n");
        getchar();
```

```
        return errorCode;
    }

    // close file and serial port
    printf("\nStopping data capture...\n");
    fclose(fp);
    CloseHandle(hSerial);

    return 0;
}

int configureSerialPort(HANDLE hSerial) {
    DCB dcbSerialParams = {0};
    COMMTIMEOUTS timeouts = {0};

    // configure serial port
    printf("Configuring serial port...\n");
    dcbSerialParams.DCBlength = sizeof(dcbSerialParams);
    if (!GetCommState(hSerial, &dcbSerialParams)) {
        printf("Error: Could not get comm state\n");
        getchar();
        return ERROR_CONFIGURE_SERIAL_PORT;
    }
    dcbSerialParams.BaudRate = CBR_9600;
    dcbSerialParams.ByteSize = 8;
    dcbSerialParams.StopBits = ONESTOPBIT;
    dcbSerialParams.Parity   = NOPARITY;
    if (!SetCommState(hSerial, &dcbSerialParams)){
        printf("Error: Could not set comm state\n");
        getchar();
        return ERROR_CONFIGURE_SERIAL_PORT;
    }

    // configure timeouts
    timeouts.ReadIntervalTimeout = 50;
    timeouts.ReadTotalTimeoutConstant = 50;
    timeouts.ReadTotalTimeoutMultiplier = 10;
    timeouts.WriteTotalTimeoutConstant = 50;
    timeouts.WriteTotalTimeoutMultiplier = 10;
    if (!SetCommTimeouts(hSerial, &timeouts)) {
        printf("Error: Could not set comm timeouts\n");
        getchar();
        return ERROR_SET_COMM_TIMEOUTS;
    }

    printf("Serial port configured successfully.\n");
    return 0;
}

int captureData(HANDLE hSerial, FILE* fp) {
```

```
    // wait for 5 seconds before starting data capture
    printf("Data capture starting in 5... ");
    Sleep(1000);
    printf("4... ");
    Sleep(1000);
    printf("3... ");
    Sleep(1000);
    printf("2... ");
    Sleep(1000);
    printf("1...\n");
    Sleep(1000);

    // flush serial port buffer
    printf("Flushing serial port buffer...\n");
    PurgeComm(hSerial, PURGE_RXCLEAR);

    // read from serial and write to file
    printf("Data capture started. Writing data to file...\nPress Ctrl+C to stop
data capture.\n\n");
    char buffer[BUFFER_SIZE];
    DWORD bytesRead;
    while (1) {
        if (ReadFile(hSerial, buffer, BUFFER_SIZE, &bytesRead, NULL)) {
            // get current date/time
            time_t t = time(NULL);
            struct tm tm = *localtime(&t);
            if ((int) bytesRead == 0) {
                continue;
            }
            // append data to file with timestamp
            fprintf(fp, "[%02d-%02d-%04d - %02d:%02d:%02d]\n%.*s", tm.tm_mday,
tm.tm_mon + 1, tm.tm_year + 1900,
                    tm.tm_hour, tm.tm_min, tm.tm_sec, (int) bytesRead, buffer);
            fflush(fp);

            // print captured data with timestamp to console
            printf("[%02d-%02d-%04d - %02d:%02d:%02d]\n%.*s", tm.tm_mday,
tm.tm_mon + 1, tm.tm_year + 1900,
                    tm.tm_hour, tm.tm_min, tm.tm_sec, (int) bytesRead, buffer);
        }
    }
}

FILE* openDataFile() {
    FILE* fp;
    char fileName[50];
    char fileFormat[10];

    // get file name and format from user
    printf("Enter file name: ");
```

```
    scanf("%s", fileName);
    while (getchar() != '\n'); // flush input buffer
    printf("Enter file format (e.g. txt): ");
    scanf("%s", fileFormat);
    while (getchar() != '\n'); // flush input buffer

    // default to ".txt" if no file format is specified or if an invalid file
format is entered
    if (strlen(fileFormat) == 0 || strlen(fileFormat) > 4) {
        strcpy(fileFormat, "txt");
    }

    // concatenate file format to file name
    strcat(fileName, ".");
    strcat(fileName, fileFormat);

    // open file for writing
    fp = fopen(fileName, "a");
    if (fp == NULL) {
        printf("Error: Could not open file for writing\n");
        return NULL;
    }

    return fp;
}
```

Com o código do script acabado passamos para o código no arduino. O código o que faz é comparar os valores e guarda-os em variáveis.

```
#include <TimerOne.h>
// prototype
void sensor_print();

// global variables
  float min_temperature = 100;
  float max_temperature = 0;
  float sum_temperature = 0;

void setup() {
  Serial.begin(9600);
  Timer1.initialize(1000000);
  Timer1.attachInterrupt(sensor_print);
}

void loop() {
  static unsigned long time = 0;
  if (millis() - time > 60000) {
    Serial.println("--------------------");
    Serial.print("min: ");
    Serial.print(min_temperature);
    Serial.println("°C");
```

```
    Serial.print("max: ");
    Serial.print(max_temperature);
    Serial.println("°C");
    Serial.print("avg: ");
    Serial.print(sum_temperature / 60);
    Serial.println("°C");
    Serial.println("--------------------");
    Serial.println();
    min_temperature = 100;
    max_temperature = 0;
    sum_temperature = 0;
    time = millis();
  }
}

void sensor_print() {
  float temperature = ((analogRead(A0) * 5.0 / 1024.0) - 0.5) * 100;
  if (temperature < min_temperature) {
    min_temperature = temperature;
  }
  if (temperature > max_temperature) {
    max_temperature = temperature;
  }
  sum_temperature += temperature;
}
```

Em python podemos ver que o código é menor e mais simplificado.

```
import serial
import time
import datetime
from tqdm import trange

WAIT_TIME = 65 # seconds

# arduino = serial.Serial(port='COM8', baudrate=9600, timeout=.1)
arduino = serial.Serial(port='COM8', baudrate=9600, timeout=.1)

def read():
    data = arduino.readline()
    return data

while True:
    value = read().decode('utf-8').strip()
    data = datetime.datetime.now().strftime("%Y-%m-%d %H:%M:%S") + " " + value
    if value != "":
        print(data)
        with open('data.txt', 'a') as f: # saving the value to a file
            f.write(value + '\n')     # saving the value to a file
```

4. Discussão

No geral, a função Timer1 é uma ferramenta poderosa para controlar eventos baseados em tempo em sistemas eletrónicos e é amplamente utilizada em projetos de automação, controle, robótica e muitas outras aplicações.
Como podemos ver no exercício 3 existiam duas formas de fazer o programa em C ou em python neste caso em concreto era mais fácil python.

5. Conclusão

Assim sendo podemos concluir que cumprimos os objetivos de todos os exercícios e conseguimos compreender o funcionamento da biblioteca TimerONE, também podemos ver que exercícios como o 3 é mais fácil utilizar python porque em C teríamos que desenvolver um outro programa, um script.

6. Referências

Fried, Limor . “Installing Libraries | Arduino Documentation.” Docs.arduino.cc, 16 May 2023, docs.arduino.cc/software/ide-v1/tutorials/installing-librarieslibraries.Accessed 25 May 2023.

“TimerOne - Arduino Reference.” Www.arduino.cc, 16 May 2023, www.arduino.cc/reference/en/libraries/timerone/.Accessed 25 May 2023.

“Windows SDK - Windows App Development.” Developer.microsoft.com, developer.microsoft.com/en-us/windows/downloads/windows-sdk.Accessed 25 May 2023.

Python. “Download Python.” Python.org, Python.org, 2019, www.python.org/downloads/.Accessed 25 May 2023.

“Windows.h.” Wikipedia, 20 Dec. 2022, en.wikipedia.org/wiki/Windows.h. Accessed 25 May 2023.

Vídeos:
https://www.youtube.com/playlist?list=PLweUC19fZUoZNt5veTkSEAhM9uARHJd29