FACULDADE: CENTRO UNIVERSITÁRIO DE BRASÍLIA – UniCEUB
CURSO: ENGENHARIA DE COMPUTAÇÃO
DISCIPLINA: SISTEMAS DE TEMPO REAL E EMBARCADOS
CARGA HORÁRIA: 60 H. A. **ANO/SEMESTRE**: 2020/02
PROFESSOR: ADERBAL BOTELHO
HORÁRIOS: Terças e Quartas às 07h40

LABORATÓRIO – SINCRONIZAÇÃO E COMUNICAÇÃO

RESUMO

Os sistemas de tempo real precisam lidar com aspectos relacionados à concorrência para aexecução de suas tarefas nos tempos especificados. O laboratório vai trabalhar alguns algoritmos para solução dos problemas relacionados à concorrência para o desenvolvimento de sistemas de tempo real.

OBJETIVOS

Objetivo Geral
Construir algoritmos para sincronização e comunicação.

Objetivos Específicos

1. Aprender a utilizar o Arduíno para implementar alguns algoritmos conhecidos;
2. Conhecer as ferramentas para realizar sincronização das tarefas;
3. Explorar teorias que poderão ser utilizadas no Projeto Final da disciplina.

EXERCÍCIO 01 – BLINK[1]

O exemplo mostra a maneira mais simples de obter resposta do Arduíno e visualizar uma resposta física: a luz pisca quando solicitada. Para implementar é necessário realizar as conexões apresentadas na Figura 1.

Para executar o tutorial serão necessários os seguintes componentes:

- Arduino
- LED
- Resistor de 220 ohm

1 Disponível em https://www.arduino.cc/en/Tutorial/Blink

EXERCÍCIO 01 – BLINK

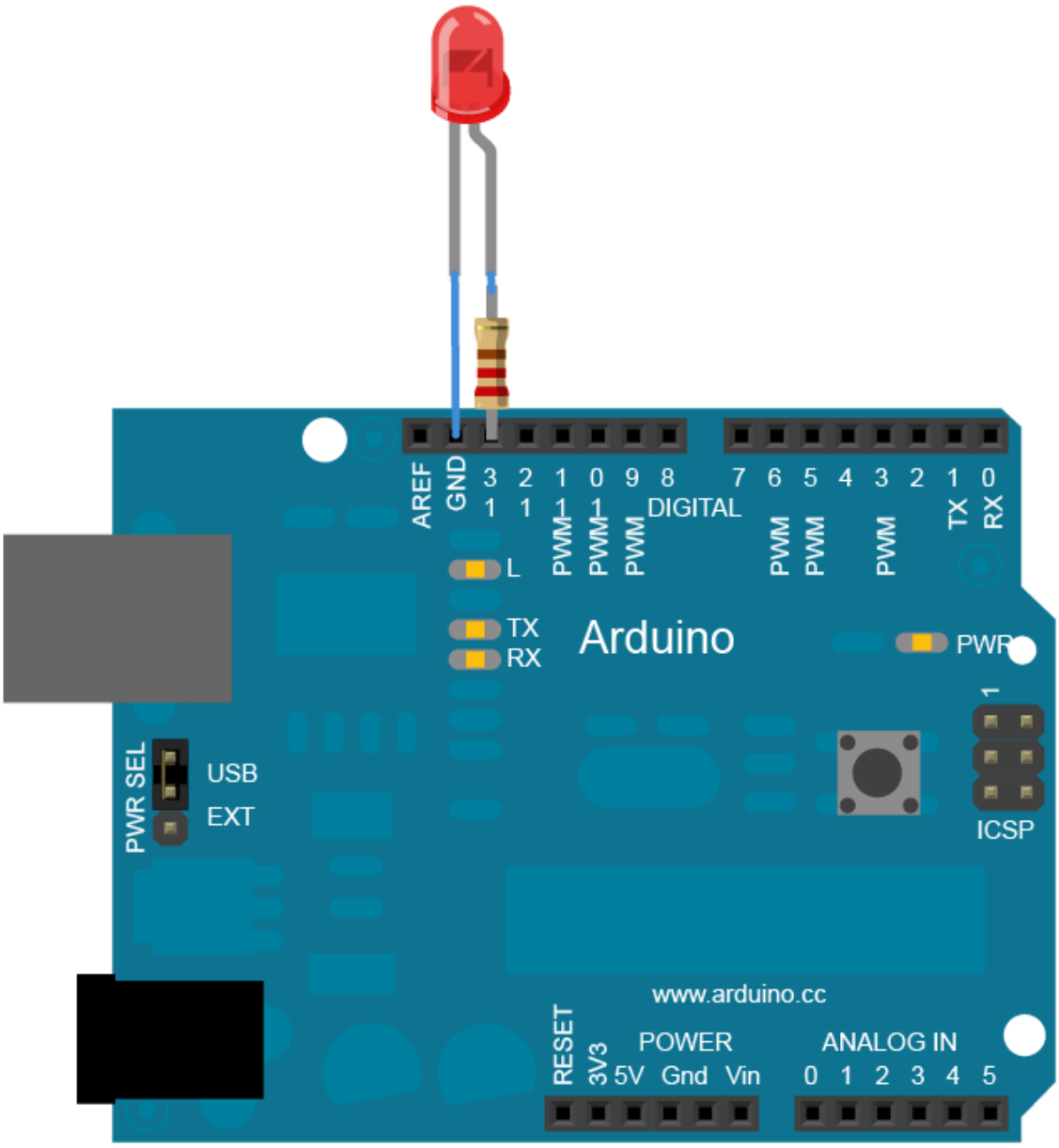

Figura 1: Esquemático do tutorial Blink

Para fazer o exemplo funcionar utilize o seguinte trecho de código:

```
/*
 Blink
 Turns on an LED on for one second, then off for one second, repeatedly.
 Most Arduinos have an on-board LED you can control. On the UNO, MEGA and ZERO
 it is attached to digital pin 13, on MKR1000 on pin 6. LED_BUILTIN is set to
 the correct LED pin independent of which board is used.
 If you want to know what pin the on-board LED is connected to on your Arduino
model, check
```

EXERCÍCIO 01 – BLINK

```
 the Technical Specs of your board at https://www.arduino.cc/en/Main/Products

 This example code is in the public domain.
 modified 8 May 2014
 by Scott Fitzgerald

 modified 2 Sep 2016
 by Arturo Guadalupi

 modified 8 Sep 2016
 by Colby Newman
*/
// the setup function runs once when you press reset or power the board
void setup() {
 // initialize digital pin LED_BUILTIN as an output.
 pinMode(LED_BUILTIN, OUTPUT);
}
// the loop function runs over and over again forever
void loop() {
 digitalWrite(LED_BUILTIN, HIGH); // turn the LED on (HIGH is the voltage
level)
 delay(1000); // wait for a second
 digitalWrite(LED_BUILTIN, LOW); // turn the LED off by making the voltage LOW
 delay(1000); // wait for a second
}
```

EXERCÍCIO 02 - DIGITALREADSERIAL[2]

O exemplo ilustra como monitorar o estado de um *switch* estabelecendo uma comunicação serial entre a placa Arduino e o seu computador através da entrada USB. O esquemático proposta está descrito na Figura 2.

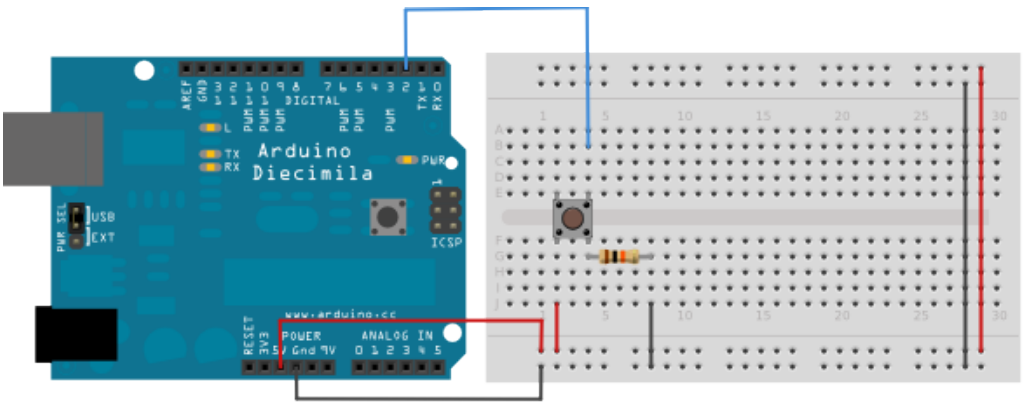

Figura 2: Esquemático para leitura do serial

2 Disponível em https://www.arduino.cc/en/Tutorial/DigitalReadSerial

EXERCÍCIO 02 - DIGITALREADSERIAL

Para executar o tutorial serão necessários os seguintes componentes:

- Placa Arduino;
- Um botão ou chave (*switch*);
- Resistor de 10k ohm;
- *Jumpers*;
- *Protoboard*.

Para fazer o exemplo funcionar utilize o seguinte trecho de código:

```
/*
 DigitalReadSerial
 Reads a digital input on pin 2, prints the result to the serial monitor
 This example code is in the public domain.
 */
// digital pin 2 has a pushbutton attached to it. Give it a name:
int pushButton = 2;
// the setup routine runs once when you press reset:
void setup() {
 // initialize serial communication at 9600 bits per second:
 Serial.begin(9600);
 // make the pushbutton's pin an input:
 pinMode(pushButton, INPUT);
}
// the loop routine runs over and over again forever:
void loop() {
 // read the input pin:
 int buttonState = digitalRead(pushButton);
 // print out the state of the button:
 Serial.println(buttonState);
 delay(1); // delay in between reads for stability
}
```

EXERCÍCIO 03 – SEMÁFOROS

Semáforos são varáveis de controle que inseridas em chamadas à região crítica indicam se a área está disponível para uso ou não. Utilizam-se de uma estrutura binária muito simples: está liberada para uso ou não está. Para que o semáforo seja consistente, a utilização da região crítica deve ser atômica, ou seja, os recursos são bloqueados durante a escrita e liberados ao final da operação.

As operações de liberação e bloqueio da região crítica são também chamadas de *down* e *up* respectivamente. Ao realizar um *down* o semáforo tem seu valor decrementado, ou seja, reduzido em 1, caso seu valor seja maior que 0. Se o valor for menor que 0, o processo é posto para dormir por enquanto. O *up* incrementa o valor em 1 e caso o valor resultante seja maior que 0 a região crítica é liberada.

EXERCÍCIO 03 – SEMÁFOROS

Para trabalhar com semáforos na plataforma Arduíno, vamos trabalhar com a biblioteca NilRTOS, que implementa algumas ferramentas úteis para sistemas de tempo real. Baixe e instale a biblioteca NilRTOS-Arduino no endereço https://github.com/greiman/NilRTOS-Arduino

EXERCÍCIO: Utilize o trecho de código a seguir para implementar um semáforo utilizando a biblioteca NilRTOS. O esquemático exige que o pino do LED seja plugado no conector digital 13, a exemplo do laboratório anterior de Arduíno.

```
/*
 * Example to demonstrate thread definition, semaphores, and thread sleep.
 */
#include <NilRTOS.h>

// The LED is attached to pin 13 on Arduino.
const uint8_t LED_PIN = 13;

// Declare a semaphore with an inital counter value of zero.
SEMAPHORE_DECL(sem, 0);
//------------------------------------------------------------------------------
/*
 * Thread 1, turn the LED off when signalled by thread 2.
 */
// Declare a stack with 128 bytes beyond context switch and interrupt needs.
NIL_WORKING_AREA(waThread1, 128);

// Declare the thread function for thread 1.
NIL_THREAD(Thread1, arg) {
  while (TRUE) {

    // Wait for signal from thread 2.
    nilSemWait(&sem);

    // Turn LED off.
    digitalWrite(LED_PIN, LOW);
  }
}
//------------------------------------------------------------------------------
/*
 * Thread 2, turn the LED on and signal thread 1 to turn the LED off.
 */
// Declare a stack with 128 bytes beyond context switch and interrupt needs.
NIL_WORKING_AREA(waThread2, 128);

// Declare the thread function for thread 2.
NIL_THREAD(Thread2, arg) {

  pinMode(LED_PIN, OUTPUT);

  while (TRUE) {
```

EXERCÍCIO 03 – SEMÁFOROS

```
    // Turn LED on.
    digitalWrite(LED_PIN, HIGH);

    // Sleep for 200 milliseconds.
    nilThdSleepMilliseconds(200);

    // Signal thread 1 to turn LED off.
    nilSemSignal(&sem);

    // Sleep for 200 milliseconds.
    nilThdSleepMilliseconds(200);
  }
}
//------------------------------------------------------------------------------
-
/*
 * Threads static table, one entry per thread.  A thread's priority is
 * determined by its position in the table with highest priority first.
 *
 * These threads start with a null argument.  A thread's name may also
 * be null to save RAM since the name is currently not used.
 */
NIL_THREADS_TABLE_BEGIN()
NIL_THREADS_TABLE_ENTRY("thread1", Thread1, NULL, waThread1, sizeof(waThread1))
NIL_THREADS_TABLE_ENTRY("thread2", Thread2, NULL, waThread2, sizeof(waThread2))
NIL_THREADS_TABLE_END()
//------------------------------------------------------------------------------
-
void setup() {
  // Start Nil RTOS.
  nilSysBegin();
}
//------------------------------------------------------------------------------
-
// Loop is the idle thread.  The idle thread must not invoke any
// kernel primitive able to change its state to not runnable.
void loop() {
  // Not used.
}
```

BIBLIOGRAFIA

SHAW, Alan C. Sistemas e Software de Tempo Real. Porto Alegre: Bookman, 2003.s

FARINES, Jean-Marie et al. Sistemas de Tempo Real. São Paulo: IME-USP, 2000. v. 1. (http://lattes.cnpq.br/4953705856223870)

IST (2009). Jantar dos filósofos em linux. Disponível em http://comp.ist.utl.pt/ec-st/Labs/Jantar-Linux.htm Acessado em 12/01/2011.