# PERCOBAAN IV
# RTOS

## 4.1 Tujuan Percobaan

Dalam bab praktikum ini diharapkan praktikan dapat menerapkan penggunaan RTOS melakukan penjadwalan multitasking

## 4.2 Perangkat yang digunakan

Perangkat:

- Perangkat lengkap Arduino UNO
- PC/Laptop yang terinstall software Arduino IDE
- LED dan resistor (220 Ohm, 10 kOhm)
- Potensiometer
- Push button
- Project board dan Kabel Jumper

## 4.3 Dasar Teori

Multitasking adalah sebuah cara untuk mengeksekusi *multiple task* atau *multiple process* dalam periode waktu tertentu. *Preemptive* dan juga *cooperative multitasking* adalah dua tipe multitasking. Dalam multitasking *preemptive*, sistem operasi dapat memulai peralihan konteks dari proses yang sedang berjalan ke proses lain. Dengan kata lain, sistem operasi memungkinkan penghentian eksekusi proses yang sedang berjalan dan mengalokasikan CPU ke beberapa proses lain. Dalam *cooperative multitasking*, sistem operasi tidak pernah memulai peralihan konteks dari proses yang sedang berjalan ke proses lain. Peralihan hanya terjadi ketika proses secara sengaja memberikan kontrol secara berkala, sedang idle, atau prosesnya terblokir untuk memungkinkan beberapa aplikasi dijalankan secara bersamaan.

### OS dan RTOS

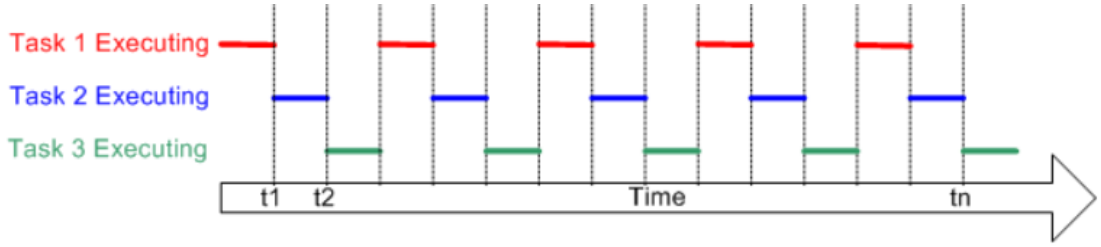
*General purpose operating system* merupakan bagian program komputer yang mendukung fungsi dasar dan menyediakan layanan untuk aplikasi lain sehingga dapat dijalankan pada komputer. Sebagai contoh, aplikasi *browser* yang berjalan di atas *environment* yang disediakan oleh OS dan tidak secara langsung berinteraksi di level hardware.

Pada kebanyakan jenis OS yang ada dapat dilakukan eksekusi multi aplikasi di waktu yang bersamaan yang disebut dengan *multitasking*. Pada kenyataannya, tiap *core* prosesor hanya dapat mengeksekusi 1 *thread* eksekusi pada satu waktu tertentu. Oleh karena itu ada bagian dari operating system yang disebut dengan *scheduler* yang bertanggung jawab menentukan program mana yang berjalan dan kapan sehingga seakan-akan berjalan secara bersamaan dengan cara berpindah eksekusi secara cepat.

*Real-time Operating System* (RTOS) adalah jenis *operating system*/sistem operasi yang memiliki 2 kunci utama yakni prediktabilitas dan deterministik. Jenis sistem operasi ini berbeda dengan sistem operasi pada smartphone atau PC/laptop. RTOS dibuat sedemikian rupa untuk bisa mengeksekusi task dengan cepat dan efektif sehingga sistem merespon sesuai dengan yang di-ekspektasi tiap saat.

*Scheduler* yang ada pada RTOS didesain untuk menyediakan pola eksekusi task yang deterministik. Pada embedded system, terkadang dijumpai *requirement* yang salah satunya adalah system harus dapat merespon pada *event* tertentu dalam durasi waktu tertentu. *Scheduler* yang digunakan pada FreeRTOS menggunakan basis *priority* yang diset oleh user pada setiap eksekusi tasknya. *Scheduler* kemudian menggunakan *priority* ini untuk memutuskan *thread/task* mana yang akan dijalankan berikutnya. FreeRTOS adalah salah satu RTOS yang didesain untuk dibuat sekecil mungkin sehingga dapat dijalankan oleh mikrokontroller. FreeRTOS menyediakan fungsi *core* dari *realtime scheduling, inter-task communication* dan sinkronisasi.

Prosesor konvensional hanya dapat menjalankan satu tugas pada satu waktu - tetapi dengan beralih antar tugas dengan cepat, sistem operasi multitugas dapat membuatnya tampak seolah-olah setiap tugas dijalankan secara bersamaan.




Gambar 1 eksekusi multitasking (freeRTOS.org)

**Semaphore**

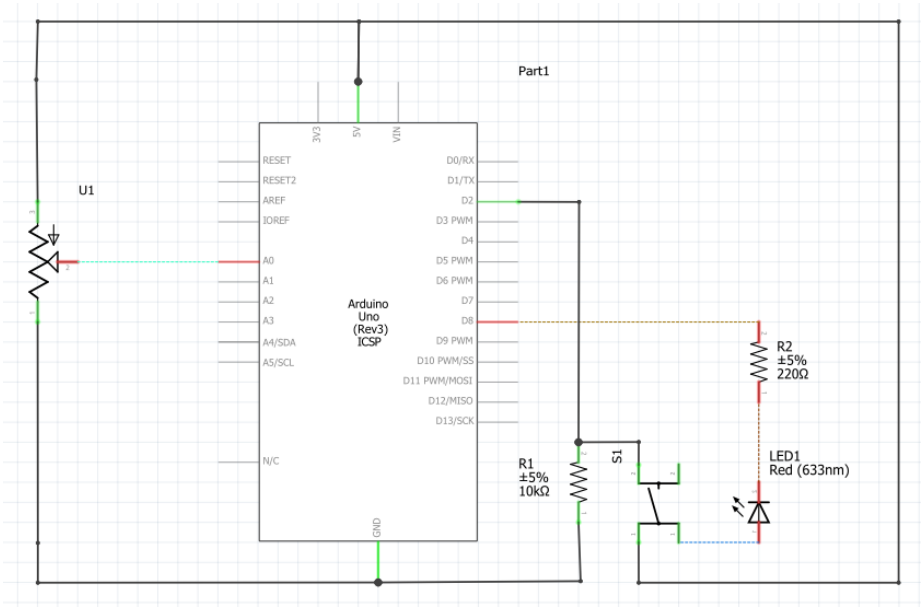
Dalam pemrograman RTOS, semaphore adalah variabel yang digunakan untuk mengontrol akses ke sumber daya bersama yang perlu diakses oleh banyak proses. Hal tersebut mirip dengan mutex yang dapat mencegah proses lain mengakses sumber daya bersama atau bagian penting. Perlu diperhatikan bahwa sumber daya bersama harus dilindungi untuk mencegah kasus overwriting. Semaphore digunakan sebagai sinyal tambahan untuk menunjukkan bahwa nilai sudah siap. Mutex, seperti yang disebutkan sebelumnya, menyebutkan bahwa task harus "mengambil" dan "memberikan" mutex, yang berarti task tersebut "memiliki" mutex selama eksekusi bagian yang *critical*.

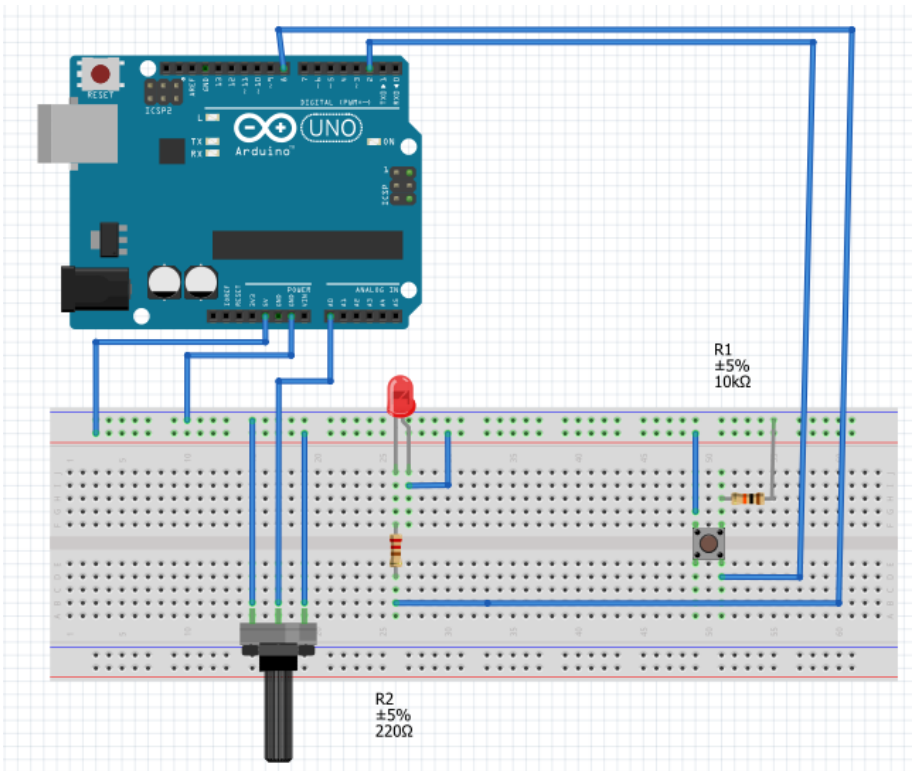
### 4.4 Prosedur percobaan

#### 4.4.1 Percobaan: Penjadwalan Multitask tanpa RTOS

1. **Prosedur Percobaan**

a. Siapkan seluruh perangkat dan komponen praktikum, lalu susun rangkaian pada *project board* sebagaimana skematik rangkaian berikut




b. Setelah rangkaian disusun dan dipastikan sesuai dengan gambar, hubungkan Arduino dengan PC/laptop melalui port USB

c. Buka Arduino IDE pada pc/laptop

**d.** Buat project baru dan masukkan *source code* berikut untuk melakukan ***multitasking* tanpa RTOS** (*comment* pada koding boleh tidak ditulis pada Arduino IDE). Lalu *compile* dan *upload* ke dalam Arduino IDE.

Keterangan: Kode di bawah ini melakukan 3 task, a) mengedipkan LED tiap detik b) membaca pin analog dan menampilkan ke serial monitor c) membaca penekanan push button dan mengedipkan LED internal sesuai penekanan

```
1. // define function
2. void TaskBlink();
3. void TaskAnalogRead();
4. void TaskDigitalRead();
5.
6. void setup() {
7.   // initialize serial communication at 9600 bits per second:
8.   Serial.begin(9600);
9.   while (!Serial) {
10.    ; // wait for serial port to connect.
11.  }
12.    pinMode(8, OUTPUT);
13.    pinMode(2, INPUT);
14.    pinMode(13, OUTPUT);
15.}
16.void loop()
17.{
18. TaskBlink();
19. TaskAnalogRead();
20. TaskDigitalRead();
21.}
22./*--------------------------------------------------*/
23./*---------------------- Function ------------------*/
24./*--------------------------------------------------*/
25.
26.void TaskBlink()
27.{
28.  /*
29.    Blink Turns on an LED on for one second, then off for one
   second, repeatedly.
30.  */
31.    digitalWrite(8, HIGH);   // turn the LED on
32.    delay(1000); // wait for one second
33.    digitalWrite(8, LOW);    // turn the LED off
34.    delay(1000); // wait for one second
35.}
36.void TaskAnalogRead()
37.{
38.  /*
39.    AnalogReadSerial
40.    Reads an analog input on pin A0, prints the result to the serial
   monitor. Graphical representation is available using serial plotter
   (Tools > Serial Plotter menu)*/
41.    // read the input on analog pin 0:
42.    int sensorValue = analogRead(A0);
43.    // print out the value you read:
44.    Serial.println(sensorValue);
45.    delay(15);  // delay (15ms) in between reads for stability
46.}
47.void TaskDigitalRead()
48.{
49.   /*
50.    DigitalRead
51.    Turns on and off a light emitting diode(LED) connected to
   digital pin 13 (onboard/built in), when pressing a pushbutton
   attached to pin 2. */
52.    // check if the pushbutton is pressed.
53.    if (digitalRead(2) == HIGH) {
54.      // turn LED on:
55.      digitalWrite(13, HIGH);
56.    } else {
```

```
57.        // turn LED off:
58.        digitalWrite(13, LOW);
59.      }
60. }
```

## 2. Hasil dan Analisis

a) Lakukan perubahan nilai input pada Arduino dengan memutar potensiometer (dan mengamati hasil pembacaannya pada serial monitor), sekaligus melakukan penekanan berulang-ulang pada push button, juga sekaligus amati output pada blinking LED

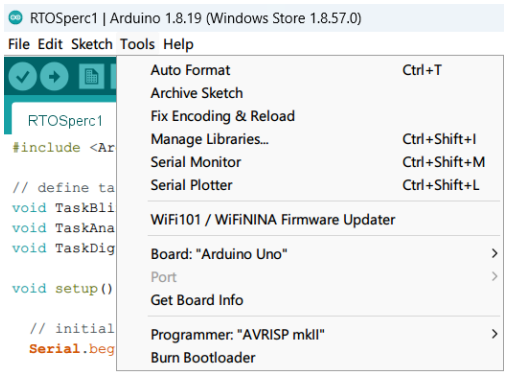
b) Jelaskan bagaimana respon masing-masing task ketika dijalankan, mengenai apakah setiap task selalu merespon tepat waktu (kedipan LED, hasil baca ADC, hasil LED sesuai penekanan push button)

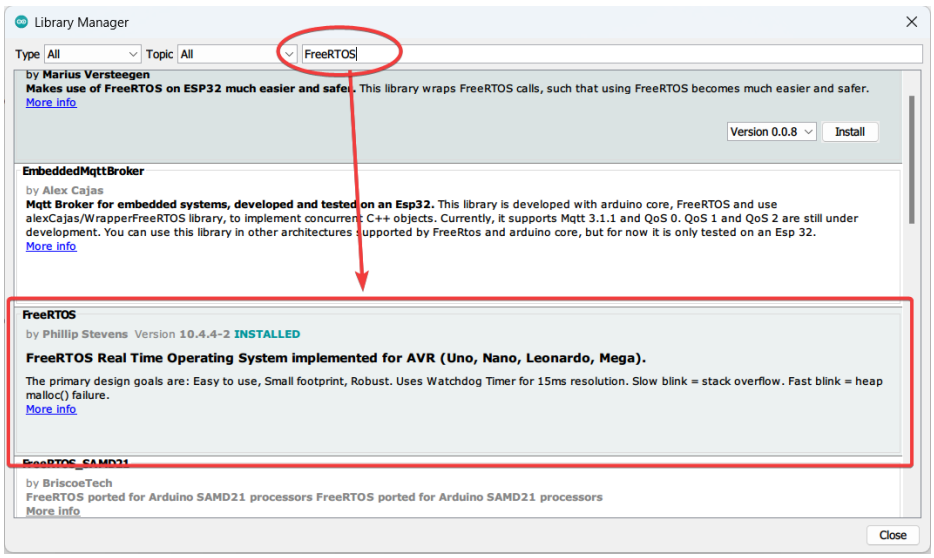
### 4.4.2 Percobaan: Penjadwalan Multasking dengan RTOS

## 1. Prosedur Percobaan

a. Susunan perangkat dan rangkaian tetap seperti pada percobaan sebelumnya

b. Lakukan instalasi library **FreeRTOS**, melalui menu **Tools>>Manage libraries..**



c. Buka project baru, masukkan *source code* berikut untuk melakukan ***multitasking* dengan RTOS.** (*comment* pada koding boleh tidak ditulis pada Arduino IDE). Lalu *compile* dan *upload* ke dalam Arduino IDE.

Keterangan: Kode di bawah ini melakukan 3 task dengan RTOS, a) mengedipkan LED tiap detik b) membaca pin analog dan menampilkan ke serial monitor c) membaca penekanan push button dan mengedipkan LED internal sesuai penekanan

```
#include <Arduino_FreeRTOS.h>
// define tasks
void TaskBlink( void *pvParameters );
void TaskAnalogRead( void *pvParameters );
void TaskDigitalRead( void *pvParameters );

void setup() {

  // initialize serial communication at 9600 bits per second:
  Serial.begin(9600);

  while (!Serial) {
    ; // wait for serial port to connect.
  }
  // Now set up tasks to run independently.
  xTaskCreate(
    TaskBlink
    ,  "Blink"   // A name just for humans
    ,  128  // stack size
    ,  NULL
    ,  2  // Priority, with 3 being the highest, 0 being the lowest.
    ,  NULL );

  xTaskCreate(
    TaskAnalogRead
    ,  "AnalogRead"
    ,  128
    ,  NULL
    ,  1  // Priority
    ,  NULL );

  xTaskCreate(
    TaskDigitalRead
    ,  "DigitalRead"
    ,  128  // Stack size
    ,  NULL
    ,  1  // Priority
    ,  NULL );

  // Now the task scheduler is automatically started.
}

void loop()
{
  // Empty. Things are done in Tasks.
}

/*--------------------------------------------------*/
/*---------------------- Tasks ---------------------*/
/*--------------------------------------------------*/

void TaskBlink(void *pvParameters)  // This is a task.
{
  (void) pvParameters;

  /*
```

```
    Blink Turns on an LED on for one second, then off for one second, repeatedly.
  */

  // initialize digital pin 8 as an output.
  pinMode(8, OUTPUT);

  for (;;) // A Task shall never return or exit.
  {
    digitalWrite(8, HIGH);   // turn the LED on
    vTaskDelay( 1000 / portTICK_PERIOD_MS ); // one second
    digitalWrite(8, LOW);    // turn the LED off
    vTaskDelay( 1000 / portTICK_PERIOD_MS ); // one second
  }
}

void TaskAnalogRead(void *pvParameters)  // This is a task.
{
  (void) pvParameters;

  /*
    AnalogReadSerial
    Reads an analog input on pin A0, prints the result to the serial monitor.
    Graphical representation is available using serial plotter (Tools > Serial Plotter menu)
  */
  for (;;)
  {
    // read the input on analog pin 0:
    int sensorValue = analogRead(A0);
    // print out the value you read:
    Serial.println(sensorValue);
    // one tick delay (15ms) in between reads for stability
    vTaskDelay(1);
  }
}

void TaskDigitalRead(void *pvParameters)  // This is a task.
{
  (void) pvParameters;
  /*
    DigitalRead
    Turns on and off a light emitting diode(LED) connected to digital pin 13 (onboard/built in), when pressing a pushbutton attached to pin 2.*/
  pinMode(2, INPUT);
  pinMode(13, OUTPUT);

        for (;;)
        {
          if (digitalRead(2) == HIGH) {
            // turn LED on:
            digitalWrite(13, HIGH);
          } else {
            // turn LED off:
            digitalWrite(13, LOW);
          }
        }
}
```

**2. Hasil dan Analisis**

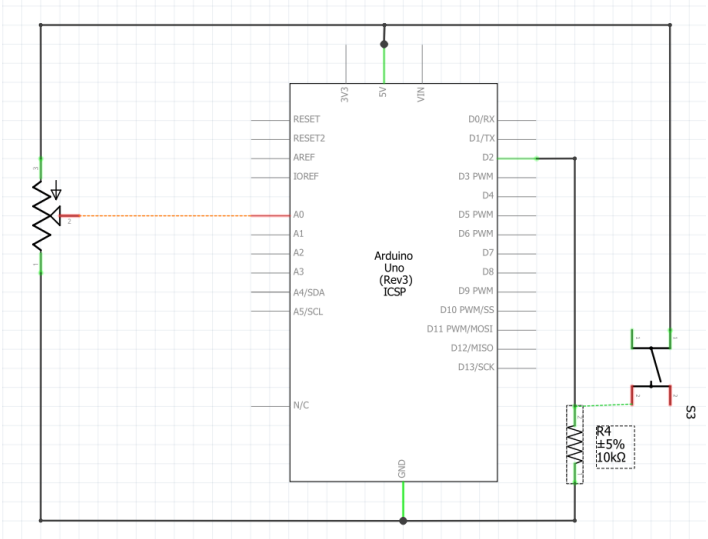
a) Lakukan perubahan nilai input pada Arduino dengan memutar potensiometer (dan mengamati hasil pembacaannya pada serial monitor), sekaligus melakukan penekanan berulang-ulang pada push button, juga sekaligus amati output pada blinking LED

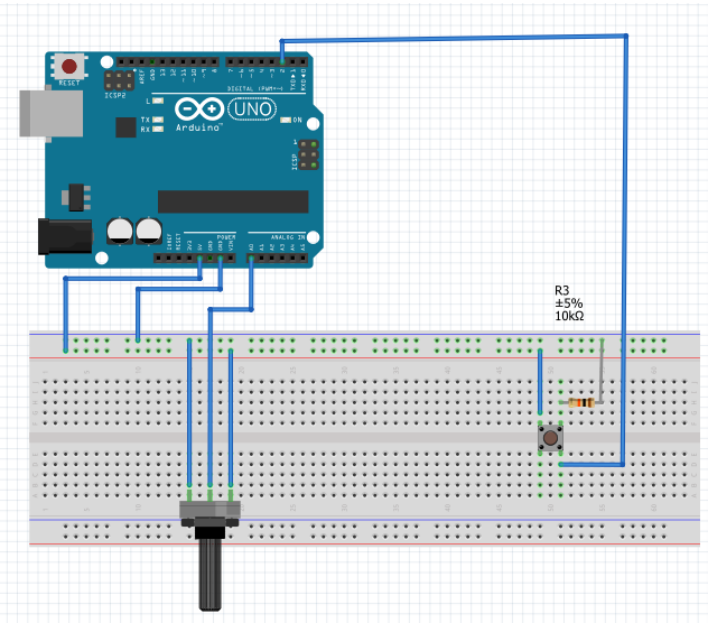
b) Jelaskan bagaimana respon masing-masing task ketika dijalankan, mengenai apakah setiap task selalu merespon tepat waktu (kedipan LED, hasil baca ADC, hasil LED sesuai penekanan push button)

### 4.4.3 Percobaan: Penggunaan Semaphore

**1. Prosedur Percobaan**

a. Siapkan seluruh perangkat dan komponen praktikum, lalu susun rangkaian pada *project board* sebagaimana skematik rangkaian berikut



b. Setelah rangkaian disusun dan dipastikan sesuai dengan gambar, hubungkan Arduino dengan PC/laptop melalui port USB

c. Buka Arduino IDE pada pc/laptop

**d.** Buat project baru dan masukkan source code berikut untuk melakukan ***multitasking*** **RTOS dengan *Semaphore*** (*comment* pada koding boleh tidak ditulis pada Arduino IDE). Lalu *compile* dan *upload* ke dalam Arduino IDE.

Keterangan: Kode di bawah ini melakukan 2 task yang mengakses *resource* Serial monitor dengan memanfaatkan *semaphore*, a) membaca pin analog dan menampilkan ke serial monitor b) membaca penekanan *push button* dan menampilkan ke serial monitor

```
#include <Arduino_FreeRTOS.h>
#include <semphr.h>  // add the FreeRTOS functions for Semaphores

/* Declare a mutex Semaphore Handle which we will use to manage the
Serial Port. It will be used to ensure only one Task is accessing
this resource at any time.*/
SemaphoreHandle_t xSerialSemaphore;

// define two Tasks for DigitalRead & AnalogRead
void TaskDigitalRead( void *pvParameters );
void TaskAnalogRead( void *pvParameters );

// the setup function runs once when you press reset or power the
board
void setup() {
  // initialize serial communication at 9600 bits per second:
  Serial.begin(9600);
  while (!Serial) {
    ; // wait for serial port to connect.
  }

  /* Semaphores are useful to stop a Task proceeding, where it
should be paused to wait, because it is sharing a resource, such as
the Serial port. Semaphores should only be used whilst the scheduler
is running, but we can set it up here.*/
  if ( xSerialSemaphore == NULL )
// confirm that the Serial Semaphore has not already been created.
  {
    xSerialSemaphore = xSemaphoreCreateMutex();
// Create a mutex semaphore we will use to manage the Serial Port
    if ( ( xSerialSemaphore ) != NULL )
      xSemaphoreGive( ( xSerialSemaphore ) );
// Make the Serial Port available for use, by "Giving" the
Semaphore.
  }

  // Now set up two Tasks to run independently.
  xTaskCreate(
    TaskDigitalRead
    ,  "DigitalRead"  // A name just for humans
    ,  128  // Stack size
    ,  NULL //Parameters for the task
    ,  2  // Priority, 3 being the highest, and 0 being the lowest.
    ,  NULL ); //Task Handle

  xTaskCreate(
    TaskAnalogRead
    ,  "AnalogRead"
    ,  128
    ,  NULL
```

```
    ,  1
    ,  NULL );

  // Now the Task scheduler is automatically started.
}

void loop()
{
  // Empty. Things are done in Tasks.
}
/*--------------------------------------------------*/
/*---------------------- Tasks ---------------------*/
/*--------------------------------------------------*/

void TaskDigitalRead( void *pvParameters __attribute__((unused)) )
// This is a Task.
{
  /*
    DigitalReadSerial
    Reads a digital input on pin 2, prints the result to the serial monitor
  */

  // digital pin 2 has a pushbutton attached to it.
  uint8_t pushButton = 2;

  // make the pushbutton's pin an input:
  pinMode(pushButton, INPUT);

  for (;;) // A Task shall never return or exit.
  {
    // read the input pin:
    int buttonState = digitalRead(pushButton);

    /* See if we can obtain or "Take" the Serial Semaphore.
    If the semaphore is not available, wait 5 ticks of the Scheduler to see if it becomes free*/
    if ( xSemaphoreTake( xSerialSemaphore, ( TickType_t ) 5 ) == pdTRUE )
    {
      /* We were able to obtain or "Take" the semaphore and can now access the shared resource. We want to have the Serial Port for us alone, as it takes some time to print, so we don't want it getting stolen during the middle of a conversion.*/
      // print out the state of the button:
      Serial.println(buttonState);
      xSemaphoreGive( xSerialSemaphore );
// Now free or "Give" the Serial Port for others.
    }
  }
}

void TaskAnalogRead( void *pvParameters __attribute__((unused)) )
// This is a Task.
{

  for (;;)
  {
    // read the input on analog pin 0:
    int sensorValue = analogRead(A0);

    /* See if we can obtain or "Take" the Serial Semaphore. If the semaphore is not available, wait 5 ticks of the Scheduler to see if it becomes free*/
    if ( xSemaphoreTake( xSerialSemaphore, ( TickType_t ) 5 ) == pdTRUE )
```

```
100.          {
101.            /* We were able to obtain or "Take" the semaphore and
    can now access the shared resource. We want to have the Serial Port
    for us alone, as it takes some time to print, so we don't want it
    getting stolen during the middle of a conversion.*/
102.            // print out the value you read:
103.            Serial.println(sensorValue);
104.            xSemaphoreGive( xSerialSemaphore );
105.       // Now free or "Give" the Serial Port for others.
106.          }
107.          vTaskDelay(1);
108.       // one tick delay (15ms) in between ADC reads for stability
109.        }
110.       }
```

**2. Hasil dan Analisis**

a) Lakukan perubahan nilai input pada Arduino dengan memutar potensiometer (dan mengamati hasil pembacaannya pada serial monitor), sekaligus melakukan penekanan berulang-ulang pada push button, dan mengamati hasil pembacaan pada serial monitor

b) Jelaskan bagaimana tampilan *resource* serial monitor dalam menangani 2 task sekaligus

### 4.5 Tugas

Buat koding (bebas) untuk melakukan penjadwalan multitask, minimal 4 task dengan periode berbeda-beda dan memanfaatkan *semaphore* untuk mengakses *resource* yang sama.

### 4.6 Kesimpulan

Berdasarkan percobaan yang telah dilakukan, maka tuliskan kesimpulan percobaan di tempat yang telah disediakan di bawah ini: