millis() tutorial

When it comes to timing things in a sketch, the first reflex is to use the delay() function. Suppose that we would like a LED to blink at a certain interval, we could create a sketch that would look like this:

```
#define LED_PIN 13;

void blink(byte ledPin, int interval) {
  digitalWrite(ledPin, HIGH);
  delay(interval);
  digitalWrite(ledPin, LOW);
  delay(interval);
}

void setup() {
  pinMode(LED_PIN, OUTPUT);
}

void loop() {
  blink(LED_PIN, 1000);
  //And do something else
}
```

This blink() function looks perfect. It can blink any pin, and at any interval. Most people would think that just adding another led, and adding another call to blink() in the loop(), both LEDs would blink together:

```
#define RED_LED_PIN 13
#define BLUE_LED_PIN 12

void blink(byte ledPin, int interval) {
  digitalWrite(ledPin, HIGH);
  delay(interval);
  digitalWrite(ledPin, LOW);
  delay(interval);
}

void setup() {
  pinMode(RED_LED_PIN, OUTPUT);
  pinMode(BLUE_LED_PIN, OUTPUT);
}

void loop() {
  blink(RED_LED_PIN, 1000);
  blink(BLUE_LED_PIN, 500);
  //And do something else
}
```

But, that is not what really happens. The red LED will blink while the blue LED stays out. Then the blue LED will blink, while the red LED stays out and repeat this cycle.

What gives?

The problem is that delay() is doing just that… delaying. When delay(1000) is called, the processor just stays there, staring at the clock, and waits for the 1000 milliseconds to pass. When the interval is finished, the processor goes on to the next instruction.

The resulting behaviour is the same as:

```
  digitalWrite(RED_LED_PIN, HIGH);
  delay(1000);
  digitalWrite(RED_LED_PIN, LOW);
  delay(1000);
  digitalWrite(BLUE_LED_PIN, HIGH);
  delay(500);
  digitalWrite(BLUE_LED_PIN, LOW);
  delay(500);
```

That is not what we wanted. If we want to succeed, we will have to find another way to blink the LEDs. That is where millis() comes in. In Arduino's guts, there is a chronometer that counts the number of milliseconds that has passed since the sketch started.

NOTE: millis() returns an `unsigned long` type value. Any variable that wants to contain what millis() returns also has to be an unsigned long. An unsigned long can count milliseconds for approximately 50 days, after which, it goes back to 0 and goes on for another 50 days cycle.

We have access to this chronometer thru millis(). This function just takes a peek at the internal chronometer and gives you what it reads. If you ask:

```
unsigned long whatTimeIsIt = millis();
```

`whatTimeIsIt` contains the number of milliseconds showed on the internal chronometer at the moment the instruction was executed.

After that, the internal chronometer will keep on ticking. If, later in the sketch we ask:

```
unsigned long whatTimeIsItNow = millis();
```

`whatTimeIsItNow` contains the number of milliseconds showed on the internal chronometer at the moment the instruction was executed. Time has passed, so the internal chronometer showed more milliseconds than the first time.

If we calculate the difference between the two readings of the internal chronometer, we will know how much time has passed between those readings.

```
Serial.println(whatTimeIsItNow – whatTimeIsIt)
```

The serial monitor will show how long it was between the first and the second time millis() was invoked.

But now, here is the big plus. Between those two calls to millis(), the processor was free to do any other tasks.

If we want to do something after a certain interval, we will need two calls to millis(). Let's name the first call previousMillis and name the second call currentMillis. Only when the difference between those two calls is at least that of the interval, will we act. That means that the second call has to be made just before we test it.

```
currentMillis = millis();
if(currentMillis - previousMillis >= interval) {
  //Do things
}
```

We do have to realise that, for an interval of 1000 milliseconds, this code will be executed millions of times before it turns out to be true and that actions will be taken inside the "if" construct.

That code is fine if we only want to time a single event. But what if we want to do is to blink the LEDs. This action has to repeat itself over and over again. The easiest way to do this is, immediately after we enter the "if" construct (the interval has now passed), to take another reading of previousMillis so it is ready to start another interval.

```
currentMillis = millis();
if(currentMillis - previousMillis >= interval) {
  previousMillis = millis();
  //Do things
```

Since currentMillis already has the information, we can write:

```
currentMillis = millis();
if(currentMillis - previousMillis >= interval) {
  previousMillis = currentMillis;
  //Do things
}
```

A third way of writing this test will eliminate the need for currentMillis (saving 4 bytes):

```
if(millis() - previousMillis >= interval) {
  previousMillis = millis();
  //Do things
```

The blinking of the LED itself can be done by remembering in which state it is right now and toggle between HIGH and LOW each time interval has reached, then write that value to the pin:

```
if (ledState == LOW) ledState = HIGH;
else                 ledState = LOW;
digitalWrite(ledPin, ledState);
```

Replacing the `//do things` part of the previous code with this gives:

```
currentMillis = millis();
if(currentMillis - previousMillis >= interval) {
  previousMillis = currentMillis;
  if (ledState == LOW) ledState = HIGH;
  else                 ledState = LOW;
  digitalWrite(ledPin, ledState);
}
```

If we only have one LED to blink, we would write:

```
const int ledPin = 13;
int ledState = 0;
unsigned long previousMillis = 0;
unsigned long interval = 1000;

void setup() {
  pinMode(ledPin, OUTPUT);
}

void loop(){
  unsigned long currentMillis = millis();
  if(currentMillis - previousMillis >= interval) {
    previousMillis = currentMillis;
    if (ledState == LOW) ledState = HIGH;
    else                 ledState = LOW;
    digitalWrite(ledPin, ledState);
}
```

But if we have many LEDs to blink, it would make sense to create a function to handle the blinking of the LEDs like we did in our first attempt.

We need to look at all the constants and variables that we used in our sketch and decide if they belong to all the LEDs or to a particular LED.

- ledPin: obviously belongs to a particular LED.
- ledState: also belongs to a particular LED.
- previousMillis: is not so obvious, but it holds the last time that the pin’s state changed. So yes, it belongs to a particular LED.
- Interval: if we want to have different intervals for each LED, then it also belongs to a particular LED.
- currentMillis: this variable can be shared by all the LEDs. (As we have seen, we could do away with this variable anyway.)

So, the first four variables all contain important information about each LED. If we have many LEDs to blink, a good way to go about it would be to use tables.

NOTE: when we declare a new table, like ledPin[4], we say that we want a table that holds 4 elements, but those elements are indexed from 0 to 3: ledPin[0], ledPin[1], ledPin[2] and ledPin[3].

Let’s suppose that our LED show needs four LEDs blinking at different rates. The tables could be:

```
byte ledPin[4] = {6, 7, 8, 9};
byte ledState[4] = {LOW, LOW, LOW, LOW};
unsigned long previousMillis[4];
int interval[4] = {1000, 500, 333, 75};
```

We can now rewrite our blink() function using our tables:

```
void blink(byte id) {                              //id will be in the range 0..3
if(millis() – previousMillis[id] >= interval) {
  previousMillis[id] = millis();
  if (ledState[id] == LOW) ledState[id] = HIGH;
  else                     ledState[id] = LOW;
  digitalWrite(ledPin[id], ledState[id]);
}
```

The sketch:

```
byte ledPin[4] = {6, 7, 8, 9};
byte ledState[4] = {LOW, LOW, LOW, LOW};
unsigned long previousMillis[4] = {0, 0, 0, 0};
int interval[4] = {1000, 500, 333, 75};

void blink(byte id) {                              //id will be in the range 0..3
if(millis() – previousMillis[id] >= interval) {
  previousMillis[id] = millis();
  if (ledState[id] == LOW) ledState[id] = HIGH;
  else                     ledState[id] = LOW;
  digitalWrite(ledPin[id], ledState[id]);
}

void setup() {
  for (byte i = 0 ; i < 4 ; i++) pinmode(ledPin[i], OUTPUT);
}

void loop() {
  for (byte i = 0 ; i < 4 ; i++) blink(i);
  //Do other stuff
}
```

I hope this tutorial helped you.
Jacques Bellavance