Arduino et « multi-taches »

Il est possible sur un Arduino de « simuler » un système multi-tache. Par exemple, on dispose de plusieurs capteurs et l'on souhaite lire leurs valeurs selon une période donnée, de plus on voudrait afficher les valeurs à une période fixée.

Bibliothèque

De nombreuses bibliothèques existent. Après plusieurs tests, il apparaît que la bibliothèque « SimpleTimer » est vraiment pratique à utiliser. Il suffit dans la procédure « setup » de déclarer les taches à exécuter à une période fixée et la procédure « loop » ne comporte qu'une seule instruction !

https://playground.arduino.cc/Code/SimpleTimer

Exemple :

```
#include <SimpleTimer.h>

SimpleTimer timer;

void tacheLireCapteur()
{
  Serial.println("Tache getLireCapteur");
}

void tacheAfficherLesValeurs()
{
  Serial.println("Tache afficherLesValeurs");
}

void setup()
{
  Serial.begin(9600);
  // Toutes les 2 secondes
  timer.setInterval(2000, tacheLireCapteur);
  // Toutes les 5 secondes
  timer.setInterval(5000, tacheAfficherLesValeurs);
}

void loop()
{
  timer.run();
}
```

Fonctions disponibles :

SimpleTimer()

The constructor. You usually need only one SimpleTimer object in a sketch.

```
SimpleTimer timer;
```

int setInterval(long d, timer_callback f)

Call function f every d milliseconds. The callback function must be declared as `void f()`.

```
void repeatMe() {
    // do something
}

timerId = timer.setInterval(1000, repeatMe);
```

int setTimeout(long d, timer_callback f)

Call function f once after d milliseconds. The callback function must be declared as `void f()`. After f has been called, the interval is deleted, therefore the value timerId is no longer valid.

```
void callMeLater() {
    // do something
}

timerId = timer.setTimeout(1000, callMeLater);
```

int setTimer(long d, timer_callback f, int n)

Call function f every d milliseconds for n times. The callback function must be declared as `void f()`. After f has been called the specified number of times, the interval is deleted, therefore the value timerId is no longer valid.

```
void repeatMeFiveTimes() {
    // do something
}

timerId = timer.setTimer(1000, repeatMeFiveTimes, 5);
```

boolean isEnabled(int timerId)

Returns true if the specified timer is enabled

```
if(timer.isEnabled(timerId) {
    // do domething
}
```

void enable(int timerId)

Enables the specified timer.

```
timer.enable(timerId);
```

void disable(int timerId)

Disables the specified timer.

```
timer.disable(timerId);
```

void toggle(int timerId)

Enables the specified timer if it's currently disabled, and vice-versa.

```
timer.toggle(timerId);
```

void restartTimer(int timerId)

Causes the specified timer to start counting from "now", i.e. the instant when restartTimer is called. The timer callback is not fired. A use case for this function is for example the implementation of a watchdog timer (pseudocode follows).

```
void wdCallback() {
    alert user or perform action to restore
    program state (e.g. reset the microprocessor)
}

wd_timer_id;

void setup() {
    wd_timer_id = timer.setInterval(10000, wdCallback);
}

void loop() {
    timer.run();

    big complex critical code

    timer.restartTimer(wd_timer_id);
}
```

void deleteTimer(int timerId)

Free the specified timerId slot. You should need to call this only if you have interval slots that you don't need anymore. The other timer types are automatically deleted once the specified number of repetitions have been executed.

void getNumTimers()

Return the number of used slots in a timer object.

```
n = timer.getNumTimers();
```