Master 1 - FST - Dpt. Info - 2015-2016
Systèmes Temps-Réel / MIF25
L. Gonnord & T. Excoffier
http://laure.gonnord.org/pro/

TP ordonnanceur Arduino

Objectifs

L'objectif de ce TP consiste à réaliser un ordonnanceur de tâches de type Round Robin. Pour simplifier le développement et mieux appréhender le concept de tâche, cet ordonnanceur fonctionnera sur une plate-forme Arduino.

Pour ce TP, le développement se fera en utilisant majoritairement le langage C.

> **Attention**, vous n'utiliserez pas la version du "language arduino" (et donc pas non plus le GUI que vous pouvez trouver sur les forums). Vous écrirez du C en utilisant l'API dont la documentation est à l'adresse :
>
> http://www.nongnu.org/avr-libc/user-manual/pages.html

Crédit J. Forget et T. Vantroys. Ce TP est adapté du sujet disponible sur cette page :

http://vantroys.polytech-lille.net/enseignement/IMA4_OS/tp_scheduler/

Au sujet de la plateforme de TP

On vous fournit par trinôme une plateforme de TP comportant 1 arduino Uno, un cable USB, une plateforme "breadboard", un écran LCD 2 lignes, deux LEDs (une rouge, une jaune), trois résistances 220 ohm, et une vingtaine de fils. **Ils seront sous votre responsabilité durant 2 semaines.**

Arduino/Atmega328p Les cartes Arduino / Guenuino sont des cartes matériellement libres architecturées autour d'un micro-contrôleur de la famille AVR d'Atmel. Pour ce TP nous utiliserons des Arduino UNO (atmega328p) comme celui de la Figure 1, qui fournissent un certain nombre d'entrées sorties numériques et analogiques sur lesquelles nous pourrons connecter des LEDs, et un écran LCD.

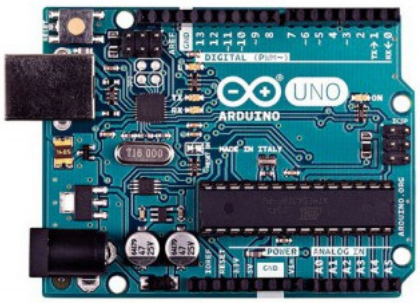

FIGURE 1 – Plateforme Arduino Uno

Le micro-contrôleur est programmé avec un *bootloader* de façon à ce qu'un programmateur dédié ne soit pas nécessaire. Les Makefile fournis utiliseront `avrdude` [1] pour charger les binaires dans la mémoire du micro-contrôleur.

Installation et manipulations préliminaires Pour pouvoir utiliser les Arduino Uno du département, il faudra :

— Installer sur les machines de TP ou sur vos machines les paquets `arduino`, `gcc-avr` et `avrdude`, et éventuellement `avr-binutils` et `avr-libc` si ils ne viennent pas avec.

— Vérifier que le fichier `avrdude.conf` est bien présent dans `/usr/share/arduino/hardware/tools/` (si il est présent ailleurs, il faudra modifier les `Makefile`).

— Permettre à l'utilisateur d'utiliser le port USB en écriture : `usermod -aG dialout <username>` (ou `useradd ...`)

Compilation et upload Dans ce TP, nous utiliserons une chaîne de compilation/assemblage/téléchargement sur la plateforme fournie par `avr-gcc` [2].

Pour compiler, vous devez utiliser le Makefile fourni. La cible all permet de compiler et la cible upload permet de transférer le binaire dans la plate-forme Arduino. Pour utiliser la cible upload sur les machines de TP, il faut au préalable brancher la plate-forme Arduino via le câble USB.

Style de programmation Les tâches exécutées sur cette plateforme seront des tâches infinies périodiques. Typiquement, un programme Arduino réalisant une tâche de période (un peu plus que) 300 ms aura la forme :

```
#include <avr/io.h>
#include <util/delay.h>
#include <avr/interrupt.h>

int main(void)
{
        //initialisation de ports, variables, ...
        while(1)
        {
          //des trucs
          _delay_ms(300);
        }
}
```

Allez, c'est parti !

1 Faire clignoter 2 LEDs

Afin de découvrir le développement sur la plate-forme Arduino, vous allez dans un premier temps réaliser un programme qui fera uniquement clignoter une LED rouge, puis une jaune. On vous fournit un ciruit déjà câblé.

1. Voir http://www.nongnu.org/avrdude/

2. Il existe une version plus *user-friendly* avec des fonctions de haut niveau et une interface graphique utilisateur, mais désirons maîtriser les temps d'exécution "cachés" derrière l'utilisation de librairies ...

1.1 Circuit

Le circuit simple représenté Figure 2 représente le câblage pour la LED rouge.

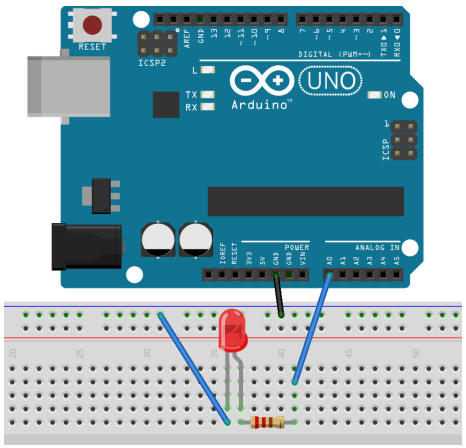

FIGURE 2 – LED sur Plateforme Arduino Uno

Notes :

— On utilise une plaque d'essai (*breadboard*) sans soudure. Dans cette plaque, tous les points d'une même ligne d'alimentation (horizontales sur le schéma) et tous les points d'une même colonne (numérotées 20, 30, ... sur le schéma) sont connectés entre eux.
— La grande patte de la LED est connectée au port Analog 0 de l'Arduino via une resistance 220 ohm.
— Le circuit est fermé en reliant la petite patte de la LED à la masse, via la ligne noire (ou bleue) du *breadboard*.

Le câblage pour la LED jaune est similaire, celle-ci sera reliée au port analog 1.

1.2 Allumage de deux LEDs

Pour allumer ou éteindre une LED, il faut configurer le registre d'entrée/sortie sur lequel est connectée la LED. Comme l'explique la documentation [3] :

— les connections "analog" 0 à 5 sont contrôlées à l'aide du registre nommé `PORTC` (commande) le bit 0 de chacun de ces deux registres commande la connection 0, le bit 1 la connection 1, ...
— le registre de configuration se nomme DDRC. Chaque bit de ce registre représente une entrée / sortie du port C (le bit 0 pour la connection 0, ...). Un 0 signifie une utilisation en tant qu'entrée et un 1 signifie une utilisation en tant que sortie.

Placez vous dans le répertoire `exo1` et éditez le fichier `main.c` (boucle infinie). En vous reportant à la documentation citée ci-dessus,

1. Écrivez la fonction `init_led` afin de pouvoir utiliser les deux LEDS. Pour activer une sortie, il faut positionner à 1 le bit correspondant du registre PORTC. [4]
2. Écrivez la boucle principale afin d'allumer la LED rouge pendant 200 ms, l'éteindre puis allumer la LED jaune pendant 300 ms puis retour à l'allumage de la LED rouge. Afin de temporiser,

3. Voir https://www.arduino.cc/en/Reference/PortManipulation. Attention, la documentation est faite pour le langage arduino de haut niveau. Pour manipuler une chaîne de bits en C on pourra utiliser `0b11111110` (à la place du `B11111110`)
4. Voir http://playground.arduino.cc/Code/BitMath si vous avez du mal avec les opérations bit à bit.

il faut utiliser la fonction `_delay_ms` qui prend en paramètre le nombre de millisecondes à attendre (et qui fait une **attente active** [5]).

3. Vérifiez le bon fonctionnement de votre programme.

2 Désynchroniser deux LEDs

Vous allez maintenant utiliser l'un des timers du micro-contrôleur pour faire clignoter les deux LEDs à deux fréquences différentes, la LED rouge clignotera toutes les 200ms et la LED jaune toutes les 400 ms en parallèle. Le montage est le même que précédemment.

Afin de faire clignoter la LED jaune toutes les 400 ms en parallèle de la LED rouge, nous allons utiliser un timer :

— Le main continue d'allumer la LED rouge comme auparavant.
— Toutes les 400 ms, une fonction d'interruption sera appelée. Elle réalisera le changement d'état de la LED jaune, puis rendra la main.

Timers et arduino L'initialisation du timer est fournie.

À chaque cycle de l'horloge, un compteur est incrémenté et il est comparé à la valeur se trouvant dans un registre (registre `OCR1A` dans notre cas). Si ils sont égaux, une interruption est générée. Le programme est alors dérouté pour exécuter la fonction correspondante et le compteur est réinitialisé [6].

Les fonctions d'interruptions, dans le cas d'un micro-contrôleur AVR et de l'utilisation de `avr-gcc` sont nommées ISR et prennent en paramètre le vecteur d'interruption correspondant (dans notre cas, il s'agit de `TIMER1_COMPA_vect`). Le timer est initialisé avec un prescaler de 1024, cela signifie que le compteur est incrémenté tous les 1024 cycles. L'arduino est cadencé à 16 MHz.

Pour activer les interruptions [7] vous devez appeler la fonction `sei()`. Pour désactiver les interruptions, la fonction est `cli()`.

Dans le fichier `main.c` du répertoire `exo2` :

1. Écrivez le code de la fonction `task_led_red` qui fait clignoter la LED rouge toutes les 200 ms (i.e., la LED est allumée pendant 200 ms, puis éteinte pendant 200 ms). Cette fonction est appelée dans la boucle de la fonction `main`. Testez le bon fonctionnement.
2. Déterminez la valeur du nombre `NB_TICK` pour initialiser le registre OCR1A, afin que la fonction d'interruption soit appelée toutes les 400 ms.
3. Remplissez la fonction ISR pour réaliser le changement d'état de la LED jaune.
4. Testez.

Grâce à l'utilisation du timer, vous avez pu réaliser ainsi deux tâches qui s'exécutent en parallèle.

3 Ordonnanceur

Vous allez maintenant réaliser un véritable ordonnanceur utilisant un algorithme *Round Robin* [8] avec un intervalle de temps de 40 ms. Votre micro-contrôleur réalisera trois tâches :

— faire clignoter la LED rouge toutes les 300 ms.

5. Voir http://www.nongnu.org/avr-libc/user-manual/group__util__delay.html, la fonction `_delay_loop_2()` utilisée fait du *busy waiting*
6. Voir par exemple http://http://www.avrbeginners.net/architecture/timers/timers.html
7. La documentation est ici http://www.nongnu.org/avr-libc/user-manual/group__avr__interrupts.html
8. Voir https://en.wikipedia.org/wiki/Round-robin_scheduling et votre cours !

— envoyer en boucle un message sur le port série (chaque envoi de caractère sera espacé de 100 ms).
— envoyer en boucle un message sur un mini écran LCD (avec une API fournie).

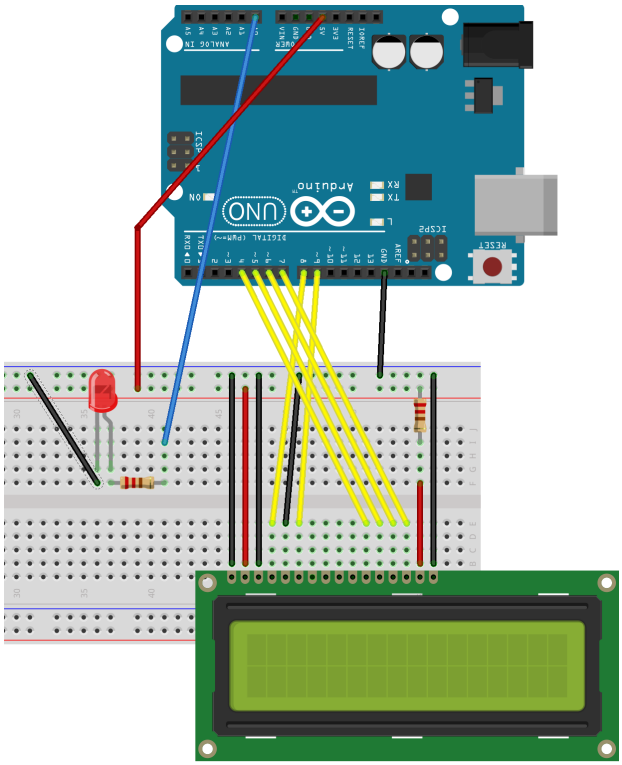

FIGURE 3 – LED et LCD sur Plateforme Arduino Uno

3.1 Circuit et configuration

On ajoute un écran LCD au circuit précédent, comme montré dans la figure 3 (il manque toujours la LED jaune) :

— Les deux LEDs sont câblées comme précédemment.
— Le LCD est relié aux ports "digital" suivant lcd(4) $\mapsto$ $D8$, lcd(5) $\mapsto$ $D9$, lcd(11) $\mapsto$ $D4$, lcd(12) $\mapsto$ $D5$, lcd(13) $\mapsto$ $D6$ lcd(14) $\mapsto$ $D7$.
— Le reste du branchement sert à assurer l'alimentation, la stabilité, et le contraste.

Écriture série, stty et screen L'écriture "port série" s'effectue en fait sur `/dev/ttyACM0`. Pour visualiser ce qui est écrit sur le port série, nous utilisons `stty` pour configurer la lecture (voir le Makefile), et le logiciel `screen`[9], dans un terminal différent de celui utilisé pour compiler/télécharger :

```
screen /dev/ttyACM0 9600
```

Pour quitter proprement screen, on fera `CTRL-a`, puis `k` (kill). Alternativement vous pouvez directement faire :

```
make && make upload && (stty 9600 ; cat) </dev/ttyACM0
```

9. Voir https://www.gnu.org/software/screen/.

3.2 Travail préliminaire

Ici chaque tâche effectuera une boucle infinie (et les `delay_ms` aussi). Le code fourni se trouve dans le répertoire `scheduler` :

1. Testez la tâche LED rouge (de période 300 ms).
2. À l'aide de la fonction `send_serial` fournie (qui envoie un caractère sur le port série), écrivez une tâche qui écrit le message de votre choix sur le port série en envoyant caractère par caractère toutes les 100 ms. Testez cette tâche.
3. Écrivez le code de la tâche écrivant une chaîne toutes les 400 ms. Vous utiliserez :
   — `lcd_write_string_4d(s)` pour imprimer une chaîne. Attention cette tâche elle même prend du temps (allez regarder le code de la librairie).
   — `lcd_write_instruction_4d(lcd_Clear)` pour effacer le texte courant.

3.3 Ordonnanceur Round Robin, sans sauvegarde de contexte

Maintenant que vos différentes tâches fonctionnent, vous pouvez réaliser l'ordonnanceur. Vous devez commencer par créer une structure (tableau de structs) représentant les différentes tâches (ici 3). Chaque structure de tâche sauvegardera ici son état (`RUNNING`, `NOT_STARTED`) :

```
typedef struct task_s {
        volatile uint16_t stack_pointer; // variable pour stocker SP
        volatile uint8_t state;         // RUNNING ou NOT_STARTED
   // vous pouvez ajouter (un pointeur vers) la fonction de la tâche
} task_t;
```

L'ordonnanceur suivra donc l'algorithme suivant :

```
  currentask <- nexttask()
  if currenttask.state == RUNNING //la tâche a déjà été interrompue
     then
          currenttask.relaunch();
     else  // premier lancement de la tâche
          currenttask.state = RUNNING ;
          sei();                 //permettre les interruptions.
          currenttask.launch();
  endif
```

Codez cet ordonnanceur (toujours dans le même répertoire) :

1. Vous utiliserez toujours un timer qui lancera une interruption régulièrement.
2. La période de l'ordonnanceur sera de 40 ms. Pour faciliter le debug, chaque fois que l'ordonnanceur s'exécute, vous allumerez la LED jaune.
3. Mettez en évidence le problème de non-restoration de contexte : les tâches reprennent toujours au début.
4. Après avoir testé, sauvez cette v1 dans un répertoire à part (`scheduler/V1/`).

3.4 Ordonnanceur Round Robin, avec sauvegarde de contexte

Pour pouvoir reprendre l'exécution d'une tâche "au bon endroit" après interruption, il faut sauvegarder "le contexte", ie l'ensemble des valeurs des registres à une certaine adresse dans la pile (SP) au moment de son interruption. On modifiera la structure de tâche en [10] :

```
typedef struct task_s {
        volatile uint16_t stack_pointer; // variable pour stocker SP.
        volatile uint8_t state; // toujours RUNNING ou NOT_STARTED
        // here you can add the associated task
} task_t;
```

Un point sur le changement de contexte Comme on l'a déjà vu en architecture, pour réaliser le changement de contexte, il est nécessaire de sauvegarder la valeur des différents registres du microcontrôleur [11].

Pour cela, on vous fournit deux macros `SAVE_CONTEXT` et `RESTORE_CONTEXT`. Pour la gestion de la mémoire des différents processus, nous allons découper l'espace mémoire. Dans un AVR atmega328p, la mémoire est utilisée en commençant par les adresses les plus élevées. La tâche "écriture série" débutera son utilisation mémoire à l'adresse 0x0700, la tâche "écriture écran LCD" débutera à l'adresse 0x0600 et la tâche "LED" débutera à l'adresse 0x0500. Ces adresses représentent le départ du pointeur de pile (Stack Pointer). Ce dernier est rendu directement accessible par `gcc-avr` via la "variable" SP.

L'ordonnanceur réalisera donc l'algorithme suivant :

```
SAVE_CONTEXT();       // sur la pile
currenttask.saveSP(); // sauver le SP dans la tâche
sei();          // permettre les interruptions
currenttask <- nexttask()
SP = currenttask.getSP();
si currenttask.state == RUNNING //la tâche a déjà été interrompue
   alors
        RESTORE_CONTEXT()
   sinon  // premier lancement de la tâche
        currenttask.state = RUNNING ;
        currenttask.launch();
finsi
```

Dans le main :

1. Observez les deux macros de stockage du contexte. Attention, dans ces macros, on appelle la fonction `cli` pour stopper les interruptions. La conséquence est qu'il faudra relancer `sei` à chaque reprise en main de l'ordonnanceur.
2. Modifier l'ordonnanceur Round Robin précédent (avec un intervalle de 40 ms) et testez-le.

4 Ressource partagée

On souhaite maintenant rajouter une tâche supplémentaire qui envoie en boucle le caractère `'@'` sur le port série. Ceci nécessite donc de gérer l'accès concurrent au port série entre deux tâches.

10. Pour une explication du volatile https://en.wikipedia.org/wiki/Volatile_(computer_programming). En a-t-on vraiment besoin ?

11. Lire aussi https://xivilization.net/~marek/binaries/multitasking.pdf

Vous allez mettre en place un mécanisme simplifié de sémaphores permettant d'assurer l'accès en exclusion mutuelle au port série.

Il faut [12] désormais trois états possibles pour une tâche : `CREATED` (la tâche n'a encore jamais été exécutée), `ACTIVE` (la tâche est disponible) et `SUSPENDED` (la tâche attend l'accès à une ressource partagée).

Après avoir réalisé une étude sérieuse du problème, copiez votre répertoire précédent sous un nouveau nom, puis :

1. Écrivez une primitive `take_serial` (similaire au P/Wait des sémaphores), qui vérifie si le port série est disponible, si oui le prend, si non la tâche l'exécutant est suspendue.
2. Écrivez une primitive `release_serial` (similaire au V/Signal des sémaphores), qui rend le port série et, si besoin, rend active la tâche suspendue.
3. Modifiez le code des tâches et de l'ordonnanceur en conséquence. Notez bien que les primitives `take_serial` et `release_serial` doivent être ininterruptibles !

5 Bonus

Pour aller plus loin, reprenez votre code dans un nouveau répertoire (appelé `bonus`) et remplacez le tourniquet par un algorithme âge et priorité. La priorité 0 est la plus élevée. L'intervalle de temps passe à 1 s. La tâche d'écriture sur le port série a la priorité 10, la tâche d'écriture sur LCD a la priorité 5 et la tâche de clignotement de la LED rouge a la priorité 2.

6 Rendu

Vous rendrez votre TP sous la forme d'une archive `tgz` sous TOMUSS au plus tard le **dimanche 8 mai 20h** [13]. Cette archive comprendra :

— Vos codes source documentés (sans produit de compilation) rangés comme dans l'archive fournie.

— Un rapport de 2 à 5 pages *ne paraphrasant pas l'énoncé*, contenant :

  — Une explication de la problématique du multitasking en AVR, et comment vous l'avez résolue à l'exercice 3.4. Vous expliquerez aussi ce que vous avez compris des *timers*.

  — L'étude sur papier de la problématique de gestion de ressources, et une explication rapide de votre implémentation.

12. En fait, dans la suite on vous donne une solution possible. Vous pouvez développer une autre solution.

13. Il va sans dire que ce travail est personnel et propre à chaque trinôme. Toute copie sera sanctionnée par un 0 non négociable.