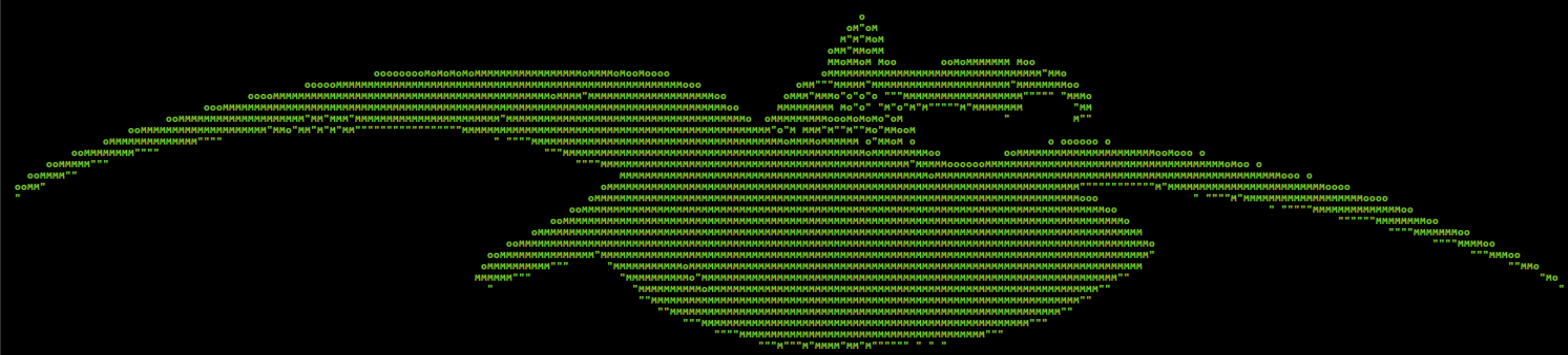
Les Universités du Tetal@b



Des Proto (pseudo) Threads pour l'Arduino

Un peu d'histoire

- Donald Ervin Knuth
  - 1973-1978 TAOCP : faire coopérer 2 unités de code plutôt que l'une contrôle l'autre par appel

- Thomas Douglas Selkirk Duff
  - 1983 Duff's device pour Lucasfilm :
    - rentrer dans une boucle et sauter sa condition avec les cases d'un switch ! (c'est possible en C)

- Simon Tatham
  - 2000 Coroutines in C

- Adam Dunkels & Oliver Schmid
  - 2005-2006 ProtoThreads

Quel est le problème ?

- Si le monde réel était séquentiel ce serait simple
- Mais, plus il y a de traitements simultanés,

  capteurs, actionneurs, écran, communications ...
- Plus le code devient compliqué

  à lire, corriger, maintenir ...
- Il faut abstraire et modulariser pour :

  cacher la complexité, avoir une vue synthétique,
  ne pas se perdre dans les détails ...
- Trouver une représentation assez bonne !

Comment exprimer le problème ?

- Machine d'état complète pour tout le processus
  - Un automate à état est souvent assez complexe
- Parallélisme réel des tâches
  - En essayant de traiter les tâches virtuellement en parallèle
- Moniteur préemptif
  - Requiert des ressources, non déterministe, problèmes des priorités
- Véritable thread
  - Utilisation de routines d'interruption
  - Requiert une bonne quantité de mémoire
- Protothreads légers
  - Pseudo threads avec commutation simplifiée

Proto Threads, pourquoi ?

- Le plus souvent le switch/case est utilisé pour réaliser une machine à états concurrents.
- Les macros des « protothread » fournissent simplement un niveau d'abstraction au dessus.
- De ce fait le code est plus linéaire et la logique d'ensemble est mieux visible

Proto Threads, comment

- C’est un genre de threads qui procure un code séquentiel en C pour réagir à des événements.
- Très légers, sans pile d’appel, avec possibilité de bloquer une exécution sans la surcharge d’une pile par fil d’exécution.
- Ce qui permet de réaliser un contrôle séquentiel sans la complexité d’un automate à états. Ils fournissent des blocages conditionnels à utiliser dans des fonctions C.

Machines & applications cibles

- Systèmes enfouis
- Systèmes à faible mémoire
- Pile de protocoles dirigés par événements
- Réseau de capteurs
- Avec ou sans moniteur temps réel (RTOS)
- Problème à tâches coopératives

Variétés de Proto Threads (PT)

- Au départ Adam Dunkels
  - Inspiré par Duff, Knuth & Tatham
  - Langage C générique
  - Permet de choisir le mécanisme de reprise,
    - « switch / case » ou « goto / label »
- Adaptations sur Arduino
  - roboticsbackend.com/wp-content/uploads/2019/06/pt.zip
  - playground.arduino.cc/Code/TimedAction
  - http://code.google.com/p/arduinode/downloads/detail?name=pt.zip

Proto Threads, propriétés

- Multi tâche coopératif
  - Utilise 2 octets / threads
- Attentes sans multi-threading
- Pas de commutation de contexte
  - Les variables ne sont normalement pas maintenu entre appel
- Portable, pur C, sans assembleur
- Utilisable avec ou sans système d'exploitation

Proto Threads, limitations

- Déclarer « static » les variables à mémoriser entre les appels
- Ne pas utiliser l'instruction « switch » au sein du protothread
- Pas d'imbrication de protothread, utiliser les macros dédiées
- Pas d'agenceur (scheduler)

Proto Threads, « hacktuce »

- Utilise les macros du préprocesseur C
  - pour cacher le « truc », la complexité & l'illisibilité !
- Une macro mémorise où il faut revenir (reprise)
  - Par n° de ligne : directive __LINE__ et instruction « switch »
  - Ou, par pointeur de label et goto
  - D'autres techniques existent : longjump, ...
- Le PT exécute du code et « return » s'il faut attendre un délai ou qu'une condition se réalise
- Au prochain appel du thread, le paramètre «reprise» permet de reprendre le traitement au point voulu

Proto Threads, code de base

- Structure de données « pt », 1 pt / protothreads
  - unsigned short lc_t ; struct pt { lc_t lc; } ;

- Macros de base
  - PT_INIT(pt)
  - PT_BEGIN(pt)
  - PT_END(pt)
  - PT_WAIT_UNTIL(pt, condition)
  - PT_WAIT_WHILE(pt, condition)
  - PT_EXIT(pt)
  - PT_YIELD(pt)

- Usage
  - Init du thread
  - Début code du thread
  - Fin code du thread
  - Attente condition
  - Tant que condition
  - Terminaison du thread
  - Sortie du thread

Le problème à 2 LEDs sans threads

```
/*  Projet: UD4S atelier thread pour l'Arduino        08/08/20  */
const byte L1=10, L2=11; // 2 leds à gérer
long former[2]={0, 0};   // mémorise la dernière commutation

void setup() {
  pinMode(L1, OUTPUT); digitalWrite(L1,HIGH);
  pinMode(L2, OUTPUT); digitalWrite(L2,HIGH);
  former[0] = former[1] = millis();
}
void loop() {
  BlinkLed(L1, 1400, former[0]);
  BlinkLed(L2, 3000, former[1]);
}
void BlinkLed (int led, int period, long& former ) {
  if ((millis() - former) >= period) {
    digitalWrite(led, !digitalRead(led));
    former = millis();
  }
}
```

Même chose avec protothreads 1/2

```
/*  Projet: UD4S atelier thread pour l'Arduino        08/08/20  */
const byte L1=10, L2=11;      // 2 leds à gérer
#include "pt.h"               // inclusion protothread

static struct pt ptL1, ptL2 ; // un par protothread
void setup() {
  pinMode(L1, OUTPUT); digitalWrite(L1,HIGH); PT_INIT(&ptL1);
  pinMode(L2, OUTPUT); digitalWrite(L2,HIGH); PT_INIT(&ptL2);
}
static PT_THREAD(ptBlink1(struct pt *pt, int period)) {
  static long former=0;       // valeur conservée entre appels
  PT_BEGIN(pt); /*-------------------- Début de code du protothread */
  While (1) {
  /* À chaque appel la condition est évaluée, si faux => return */
    PT_WAIT_UNTIL(pt, millis() - former >= period );
    digitalWrite(L1, !digitalRead(L1));
    former = millis(); }
  PT_END(pt);  /*--------------------- Fin de code du protothread */
}
```

Même chose avec protothreads 2/2

```
static PT_THREAD(ptBlink2(struct pt *pt, int period)) {
  /*------ code analogue à ptBlink1 ------*/
}
/* Boucle traitement, on lance chaque protothread avec ses paramètres */
void loop() {
  ptBlink1(&ptL1, 1400);
  ptBlink2(&ptL2, 3000);
}
```

- Chaque fil/thread est isolé dans une fonction
- Code plus verbeux, mais complexité cachée
- Code plus facile à faire évoluer :
  - L'ajout d'un thread c'est juste une fonction de plus

« #include "pt.h" » les inits

```
typedef unsigned short lc_t;
#define LC_INIT(s) s = 0; /* LC_INIT */
#define LC_RESUME(s) switch(s) { case 0:  /* LC_RESUME  */
#define LC_SET(s) s = __LINE__; case __LINE__: /* LC_SET line */
#define LC_END(s) }  /* LC_END */
struct pt {
  lc_t lc;
};
#define PT_WAITING 0
#define PT_YIELDED 1
#define PT_EXITED  2
#define PT_ENDED   3
#define PT_INIT(pt)   LC_INIT((pt)->lc) /* PT_INIT */
```

- « lc » (pour « local continuation ») contiendra la ligne de départ du thread, et c’est ce nombre qui permet ensuite de revenir dans le bon case du « switch » crée dans ce thread

« #include "pt.h" » le reste

```
#define PT_THREAD(name_args) char name_args  /* PT_THREAD */

#define PT_BEGIN(pt)                                    \
   { char PT_YIELD_FLAG = 1; LC_RESUME((pt)->lc)  /* PT_BEGIN */

#define PT_WAIT_UNTIL(pt, condition)                    \
  do {                                                  \
    LC_SET((pt)->lc);                                   \
    if(!(condition)) {                                  \
      return PT_WAITING;                                \
    }                                                   \
  } while(0)  /* PT_WAIT_UNTIL */

#define PT_END(pt) LC_END((pt)->lc); PT_YIELD_FLAG = 0; \
  PT_INIT(pt); return PT_ENDED; }  /* PT_END */
```

- Pour cet exemple on regroupe la lib dans un seul .h, agrémenté de commentaires, et on se limite au 3 macros de base avec PT_WAIT_UNTIL.

Que voit-on après l'inclusion ?

- Passons le préprocesseur :
  - avr-cpp -E -CC blink_pt.ino -o blink_pt.pp
  - Avec :
    - « -E » préprocesseur seul (sans compil ni link)
    - « -CC » garder les commentaires du pt.h
- Code obtenu après substitution de PT_INIT :

```
void setup() {
  pinMode(L1, OUTPUT); digitalWrite(L1,HIGH);
  pinMode(L2, OUTPUT); digitalWrite(L2,HIGH);
  (&ptL1)->lc = 0; /* LC_INIT */ /* PT_INIT */;
  (&ptL2)->lc = 0; /* LC_INIT */ /* PT_INIT */;
}
```

- PT_INIT « appelle » LC_INIT qui initialise lc à 0

Que voit-on après l'inclusion ?

```
static char ptBlink1(struct pt *pt, int period) /* PT_THREAD*/ {
  static long former=0;
  { char PT_YIELD_FLAG = 1; switch((pt)->lc) { case 0: /* LC_RESUME*/
/* PT_BEGIN*/;
  while (1) {
    do { (pt)->lc = 18; case 18: /* LC_SET line*/; if(!(millis() - former >=
period)) { return 0; } } while(0) /* PT_WAIT_UNTIL*/;
    digitalWrite(L1, !digitalRead(L1));
    former = millis();
  }
  } /* LC_END*/; PT_YIELD_FLAG = 0; (pt)->lc = 0; /* LC_INIT*/ /* PT_INIT*/;
return 3; } /* PT_END*/;
}
```

- PT_BEGIN crée un « switch » avec le 1er « case 0 »
- PT_WAIT_UNTIL crée un « do while(0) » avec le « case 18 » de reprise, fait avec le n° de ligne, puis renvoie 0 si condition vraie
- PT_END ferme le « switch » et renvoie 3 (« PT_ENDED »)

«Do while» imbriqué dans le «Switch»

```
static char ptBlink1(struct pt *pt, int period) /* PT_THREAD*/ {
  static long former=0;
  { char PT_YIELD_FLAG = 1;
    switch((pt)->lc) /* ou va-t-on ? */
      {
        case 0: /* LC_RESUME*/ /* PT_BEGIN 1er passage */;
        while (1) { /* while vrai */
            do {
                (pt)->lc = 18; /* note le n° de line */
        case 18: /* LC_SET line*/ ; /* case de reprise */
                  if(!(millis() - former >= period)) {
                    return 0 ; } /* condition fausse */
            } while(0) /* PT_WAIT_UNTIL */; /* do while faux */
            digitalWrite(L1, !digitalRead(L1)); /* condition vraie */
            former = millis();
        } /* while (1) */
      } /* LC_END ferme la "{" du switch */;
    PT_YIELD_FLAG = 0; (pt)->lc = 0; /* LC_INIT*/ /* PT_INIT*/;
    return 3;
  } /* PT_END  ferme la 1ère "{" du PT_BEGIN */ ;
}
```

SWITCH (lc)

DO WHILE (FAUX)

Après le 1er appel et pour cet exemple on revient toujours ici

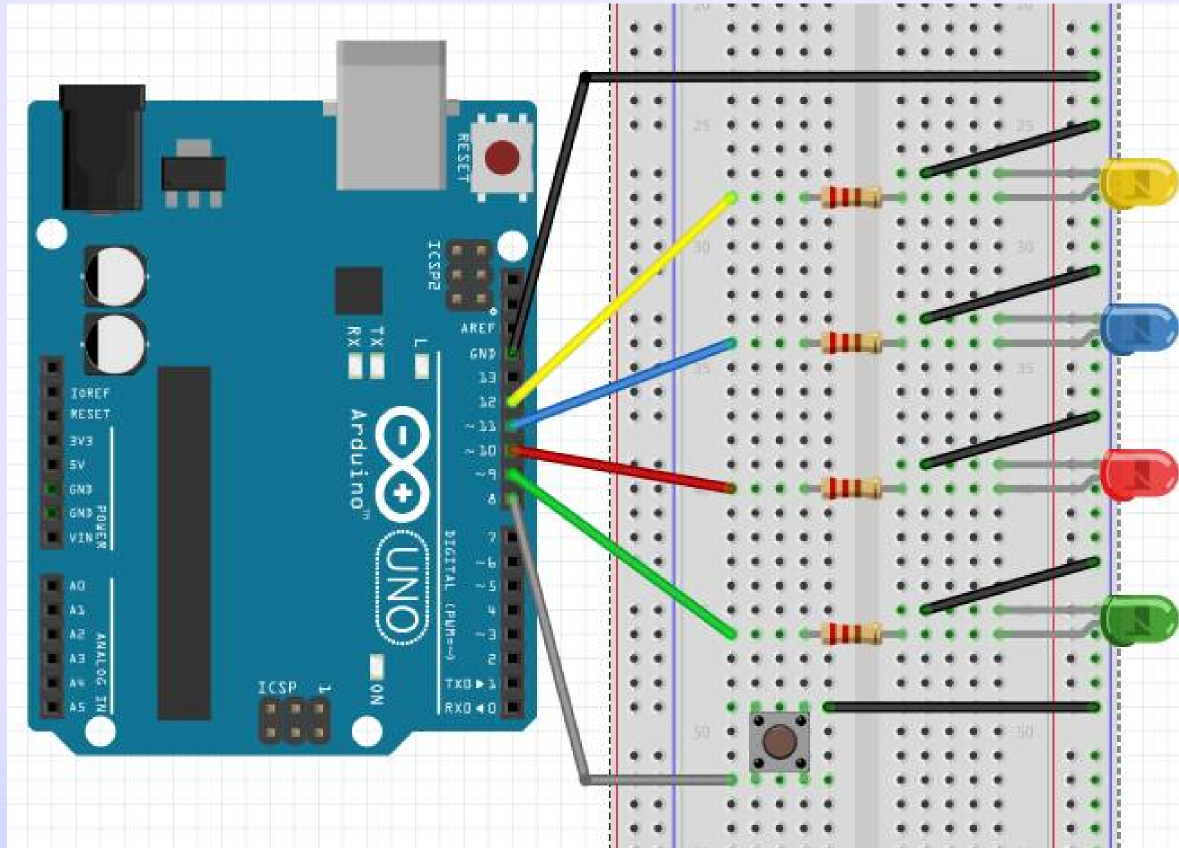
Pas de soucis ! On va mettre en pratique pour bien capter !

Mise en pratique 1

- Gérer 4 Leds cycliquement :
  - 3 des leds sont activées par un poussoir, avec chacune son top de départ et sa durée dans le cycle.
  - La 1ère, à l'appui du poussoir, a un allumage progressif, un palier, une extinction, une attente, puis recommence le cycle.
  - La 2ème et la 3éme s'allument un certain temps à un instant du cycle tant que le mode dégradé n'est pas actif.
  - La 4ème commence à 100 % et baisse de luminosité à chaque cycle. Après, N cycles le système passe en mode dégradé, inhibe les leds 2 et 3 et la led 4 clignote rapidement.

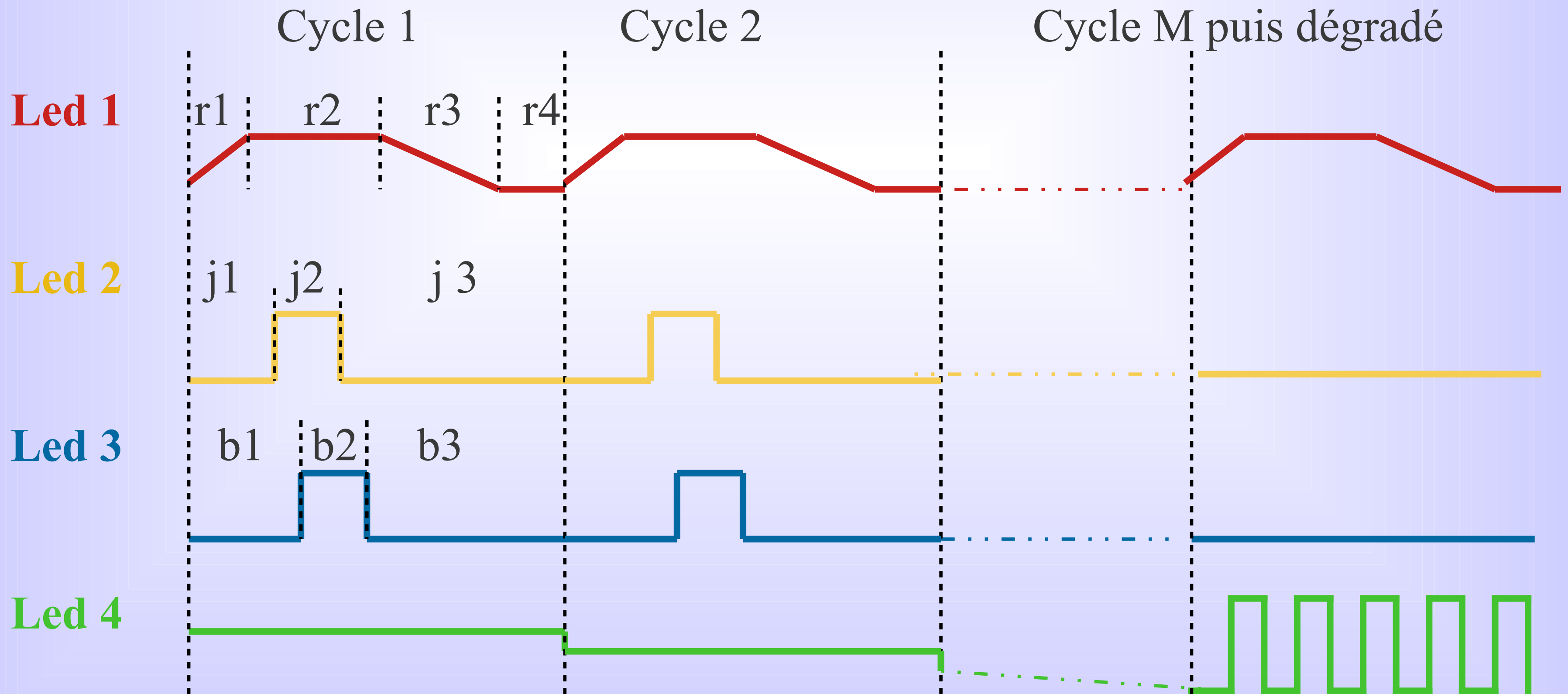
Mise en pratique 2

- Montage :



Mise en pratique 3

- Chronogramme
  - t0 à l'appui du poussoir



Mise en pratique 4

- Consignes :
  - le compteur de cycle est initialisé à M puis décrémenté à chaque appui sur le bouton poussoir
  - **Led1** PWM passe du niveau PWM 0 à 254 sur r1, puis allumée sur r2, passe de PWM 254 à 0 sur r3, éteinte sur r4
  - **Led2** éteinte sur j1, allumée sur j2, éteinte sur j1
  - **Led3** éteinte sur b1, allumée sur b2, éteinte sur b3
  - **Led4** PWM 254 au cycle 1 et passe à PWM 0 quand le compteur est null, puis clignote à 1hz en mode dégradé.

Mise en pratique 5

- N'activez pas l'IDE Arduino, cogitez d'abord !
  - Comment synchroniser les leds 1 à 3 ?
  - Comment faire correspondre la gradation de la led 1 sur les temps qui lui sont impartis ?
  - Quels paramètres vont contrôler chaque led ?
  - Quelles sont les constantes / paramètres globeaux ?
  - Combien de Protothreads ?
  - Quelles dépendances entre eux ?
  - Qu'est-ce qui n'est pas clair ?
- C'est bon ?

Mise en pratique 6

- Comment synchroniser les leds & quels sont les valeurs globales ?
  - une variable logique globale valide le début de cycle,
    un entier initialisé avec la durée du cycle, et les PT attendent le début du cycle et font ce qu'ils ont à faire durant la durée du cycle.
  -
  - Comment faire correspondre la gradation de la led 1 aux temps qui lui sont impartis ?
  - Quels paramètres vont contrôler chaque led ?
  - Combien de Proto Threads ?
  - Quelles dépendances entre eux ?
  - Qu'est-ce qui n'est pas clair ?
- C'est bon ?

Mise en pratique 6

Mise en pratique