

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 rendre modulaire pour
 - Cacher la complexité.
 - Avoir une vue synthétique.
- ◆ Et trouver une représentation assez bonne !

Comment exprimer le problème où des processus se partagent un seul MCU ?

- ◆ Machine d'état complète pour tout le processus
 - Un automate à état c'est relativement complexe.
- ◆ Parallélisme réel des tâches
 - Multiprocesseur ou virtuellement en parallèle.
- ◆ Moniteur préemptif
 - Requier des ressources, non déterministe, problèmes des priorités.
- ◆ Véritable thread
 - Utilisation de routines d'interruption.
 - Requier une bonne quantité de mémoire.
- ◆ Protothreads légers
 - Pseudo threads avec commutation simplifiée.

Les Proto Threads (ou promofilets)

- ◆ Objectif
 - Simuler du parallélisme sur un seul processeur.
- ◆ Usage
 - On écrit les traitements sans les séquencer.
 - Chaque traitement a des conditions d'exécution.
 - Un traitement bloqué passe le CPU à d'autres.
- ◆ Résultat
 - Code conduit par les évènements.
 - Évite la complexité d'un automate à états.
 - Pas besoin d'un moniteur temps réel.

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 généré par le préprocesseur (cpp) .
- Permet de choisir le mécanisme de réactivation,
 - « switch / case » ou « goto / label ».

◆ Adaptations sur Arduino

- <https://dunkels.com/adam/pt/>
- <https://roboticsbackend.com/arduino-prototreads-tutorial>
- <http://code.google.com/p/arduino/downloads/detail?name=pt.zip>

Les Proto Threads de Dunkels

- ◆ 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
- ◆ Avec ou sans système d'exploitation
- ◆ Exploite l'imbrication do/case autorisée en C

« L'hacktuce » à Dunkels

- ◆ 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.
- ◆ À l'appel suivant du thread, le paramètre «reprise» permet de reprendre le traitement au point voulu.

« L'hacktuce » à Dunkels, suite

- ◆ Exploite la particularité du C
 - Le C permet d'avoir une boucle dans un SWITCH où cette boucle contient plusieurs CASE.
 - Et le SWITCH va pouvoir adresser un CASE dans la boucle.
- ◆ Si le SWITCH constitue le corps d'une fonction
 - Ceci permet de progresser dans la boucle à chaque appel.
 - Le pré-processeur renseigne les CASE avec leur n° de ligne.
 - Et on déclare comme proto thread un pointeur sur entier.
- ◆ À l'appel suivant le SWITCH
 - Utilise cet entier pour aller au dernier traitement en cours.
- ◆ Si le traitement est en attente, on quitte la fonction

Proto Threads, en C

- ◆ Structure de données « pt », 1 pt / proto threads
 - unsigned short lc_t ; struct pt { lc_t lc; } ; // un entier
- ◆ 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

«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 */ ;
  }
```

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

SWITCH (lc)

DO WHILE (FAUX)

Proto Threads, limitations

- ◆ Dans la fonction du PT déclarer « static » les variables à mémoriser entre les appels.
- ◆ Ne pas utiliser l'instruction « switch » au sein du proto thread.
- ◆ Pas d'imbrication de proto thread, utiliser les macros dédiées.
- ◆ Pas d'agenceur (scheduler).

En résumé, les Proto Threads

- ◆ Très légers, avec juste un index qui mémorise où revenir dans le code entre chaque activation.
- ◆ Les macros des « proto thread » fournissent simplement un niveau d'abstraction au dessus.
- ◆ De ce fait le code est plus linéaire et la logique d'ensemble est mieux visible.

Le problème à 2 LEDs sans threads

Tous ces exemples fonctionnent avec l'IDE Arduino legacy 1.8.19

```
/*  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 proto threads 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 proto thread

static struct pt ptL1, ptL2 ; // un par proto thread
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 proto thread */
    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 proto thread */
}
```

Même chose avec proto threads 2/2

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

- ◆ Chaque fil/thread est isolé dans une fonction.
- ◆ Ici code plus verbeux, mais complexité cachée.
- ◆ Code plus facile à faire évoluer :
 - ajouter un thread c'est ajouter une fonction.

« #include "pt.h" » les init

```
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 : (.../arduino-1.8.19/hardware/tools/avr/bin)
 - `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 1^{er} « 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 »).

That's all Folks!



La suite au prochain épisode