TP MI11 - Réalisation d'un mini noyau temps réel ARM - Parties 1 et 2

Théophile DANCOISNE et Louis FRERET

Mai 2017

```
/* NOYAU.C */
/*-------------------------------------------------------------------------*
 *               Code C du noyau preemptif qui tourne sur ARM               *
 *                                NOYAU.C                                  *
 *-------------------------------------------------------------------------*/

#include <stdint.h>

#include "serialio.h"
#include "imx_timers.h"
#include "imx_aitc.h"
#include "noyau.h"

/*-------------------------------------------------------------------------*
 *            Variables internes du noyau                                  *
 *-------------------------------------------------------------------------*/
static int compteurs[MAX_TACHES]; /* Compteurs d'activations */
CONTEXTE _contexte[MAX_TACHES];   /* tableau des contextes */
volatile uint16_t _tache_c;       /* num?ro de tache courante */
uint32_t _tos;                   /* adresse du sommet de pile */
int _ack_timer = 1;              /* = 1 si il faut acquitter le timer */

/*-------------------------------------------------------------------------*
 *                        Fin de l'execution                               *
 *-------------------------------------------------------------------------*/
void noyau_exit(void) {
  int j;
  _irq_disable_();                /* D?sactiver les interruptions */
  printf("Sortie du noyau\n");
  for (j = 0; j < MAX_TACHES; j++)
    printf("\nActivations tache %d : %d", j, compteurs[j]);
  for (;;);                       /* Terminer l'ex?cution */
}

/*-------------------------------------------------------------------------*
 *                        --- Fin d'une tache ---                           *
 * Entree : Neant                                                          *
 * Sortie : Neant                                                          *
 * Descrip: Cette proc. doit etre appelee a la fin des taches              *
 *                                                                         *
 *-------------------------------------------------------------------------*/
void fin_tache(void) {
  /* on interdit les interruptions */
  _irq_disable_();
  /* la tache est enlevee de la file des taches */
  _contexte[_tache_c].status = CREE;
  retire(_tache_c);
  schedule();

}

/*-------------------------------------------------------------------------*
```

```
53  *                          --- Creer une tache ---                          *
54  * Entree : Adresse de la tache                                              *
55  * Sortie : Numero de la tache creee                                         *
56  * Descrip: Cette procedure cree une tache en lui allouant                   *
57  *    - une pile                                                             *
58  *    - un numero                                                            *
59  * Err. fatale: priorite erronnee, depassement du nb. maximal de taches      *
60  *                                                                           *
61  *---------------------------------------------------------------------------*/
62 uint16_t cree(TACHE_ADR adr_tache) {
63   CONTEXTE *p;                    /* pointeur d'une case de _contexte */
64   static uint16_t tache = -1;   /* contient numero dernier cree */
65 
66 
67   _lock_();                       /* debut section critique */
68   tache++;                        /* numero de tache suivant */
69 
70   if (tache >= MAX_TACHES)        /* sortie si depassement */
71     noyau_exit();
72 
73   p = &_contexte[tache];          /* contexte de la nouvelle tache */
74 
75   p->sp_ini = _tos;               /* allocation d'une pile a la tache */
76   _tos -= PILE_TACHE + PILE_IRQ;  /* decrementation du pointeur de pile pour*/
77   /* la prochaine tache. */
78 
79   _unlock_();                     /* fin section critique */
80 
81   p->tache_adr = adr_tache;       /* memorisation adresse debut de tache */
82   p->status = CREE;                  /* mise a l'etat CREE */
83   return (tache);                  /* tache est un uint16_t */
84 }
85 
86 /*---------------------------------------------------------------------------*
87  *                         --- Elire une tache ---                          *
88  * Entree : Numero de la tache                                               *
89  * Sortie : Neant                                                            *
90  * Descrip: Cette procedure place une tache dans la file d'attente des       *
91  *      taches eligibles.                                                    *
92  *      Les activations ne sont pas memorisee                                *
93  * Err. fatale: Tache inconnue                                               *
94  *                                                                           *
95  *---------------------------------------------------------------------------*/
96 void active(uint16_t tache) {
97   CONTEXTE *p = &_contexte[tache]; /* acces au contexte tache */
98 
99   if (p->status == NCREE)
100    noyau_exit();                  /* sortie du noyau */
101 
102  _lock_();                       /* debut section critique */
103  if (p->status == CREE)          /* n'active que si receptif */
104  {
105    p->status = PRET;             /* changement d'etat, mise a l'etat PRET */
106    ajoute(tache);                /* ajouter la tache dans la liste */
107    schedule();                /* activation d'une tache prete */
108  }
109  _unlock_();                     /* fin section critique */
110 }
111 
112 /*---------------------------------------------------------------------------*
113  *                  ORDONNANCEUR preemptif optimise                          *
114  *                                                                           *
115  *             !! Cette fonction doit s'ex?cuter en mode IRQ !!              *
116  *  !! Pas d'appel direct ! Utiliser schedule pour provoquer une             *
117  *  commutation !!                                                           *
118  *---------------------------------------------------------------------------*/
119 void __attribute__((naked)) scheduler(void) {
120   register CONTEXTE *p;
```

```
121   register unsigned int sp asm("sp");  /* Pointeur de pile */
122
123   /* Sauvegarder le contexte complet sur la pile IRQ */
124   __asm__ __volatile__(
125   "stmfd  sp, {r0-r14}^\t\n"  /* Sauvegarde registres mode system */
126       "nop\t\n"                 /* Attendre un cycle */
127       "sub    sp, sp, #60\t\n"    /* Ajustement pointeur de pile */
128       "mrs    r0, spsr\t\n"       /* Sauvegarde de spsr_irq */
129       "stmfd  sp!, {r0, lr}\t\n");/* et de lr_irq */
130
131   if (_ack_timer)                 /* R?initialiser le timer si n?cessaire */
132   {
133     register struct imx_timer *tim1 = (struct imx_timer *) TIMER1_BASE;
134     tim1->tstat &= ~TSTAT_COMP;
135   } else {
136     _ack_timer = 1;
137   }
138
139   _contexte[_tache_c].sp_irq = sp;/* memoriser le pointeur de pile */
140   _tache_c = suivant();           /* recherche du suivant */
141   if (_tache_c == F_VIDE) {
142     printf("Plus rien ? ordonnancer.\n");
143     noyau_exit();                 /* Sortie du noyau */
144   }
145   compteurs[_tache_c]++;          /* Incr?menter le compteur d'activations  */
146   p = &_contexte[_tache_c];       /* p pointe sur la nouvelle tache courante*/
147
148   if (p->status == PRET)          /* tache prete ? */
149   {
150     sp = p->sp_ini;               /* Charger sp_irq initial */
151     _set_arm_mode_(ARMMODE_SYS);  /* Passer en mode syst?me */
152     sp = p->sp_ini - PILE_IRQ;    /* Charger sp_sys initial */
153     p->status = EXEC;             /* status tache -> execution */
154     _irq_enable_();               /* autoriser les interuptions   */
155     (*p->tache_adr)();            /* lancement de la t?che */
156   } else {
157     sp = p->sp_irq;               /* tache deja en execution, restaurer sp_irq */
158   }
159
160   /* Restaurer le contexte complet depuis la pile IRQ */
161   __asm__ __volatile__(
162   "ldmfd  sp!, {r0, lr}\t\n"  /* Restaurer lr_irq */
163       "msr    spsr, r0\t\n"       /* et spsr_irq */
164       "ldmfd  sp, {r0-r14}^\t\n"  /* Restaurer registres mode system */
165       "nop\t\n"                   /* Attendre un cycle */
166       "add    sp, sp, #60\t\n"    /* Ajuster pointeur de pile irq */
167       "subs   pc, lr, #4\t\n");   /* Retour d'exception */
168 }
169
170 /*-------------------------------------------------------------------------*
171  *                  --- Provoquer une commutation ---                      *
172  *                                                                         *
173  *             !! Cette fonction doit s'ex?cuter en mode IRQ !!            *
174  *  !! Pas d'appel direct ! Utiliser schedule pour provoquer une           *
175  *  commutation !!                                                         *
176  *-------------------------------------------------------------------------*/
177 void schedule(void) {
178   _lock_();                       /* Debut section critique */
179
180   /* On simule une exception irq pour forcer un appel correct ? scheduler().*/
181   _ack_timer = 0;
182   _set_arm_mode_(ARMMODE_IRQ);      /* Passer en mode IRQ */
183   __asm__ __volatile__(
184   "mrs  r0, cpsr\t\n"           /* Sauvegarder cpsr dans spsr */
185       "msr  spsr, r0\t\n"
186       "add  lr, pc, #4\t\n"         /* Sauvegarder pc dans lr et l'ajuster */
187       "b    scheduler\t\n"          /* Saut ? scheduler */
188   );
```

```
189   _set_arm_mode_(ARMMODE_SYS);       /* Repasser en mode system */
190
191   _unlock_();                        /* Fin section critique */
192 }
193
194 /*-------------------------------------------------------------------------*
195  *                        --- Lancer le systeme ---                        *
196  * Entree : Adresse de la premiere tache a lancer                          *
197  * Sortie : Neant                                                          *
198  * Descrip: Active la tache et lance le systeme                            *
199  *                                                                         *
200  *                                                                         *
201  * Err. fatale: Neant                                                      *
202  *                                                                         *
203  *-------------------------------------------------------------------------*/
204 void start(TACHE_ADR adr_tache) {
205   short j;
206   register unsigned int sp asm("sp");
207   struct imx_timer *tim1 = (struct imx_timer *) TIMER1_BASE;
208   struct imx_aitc *aitc = (struct imx_aitc *) AITC_BASE;
209
210   for (j = 0; j < MAX_TACHES; j++) {
211     _contexte[j].status = NCREE;     /* initialisation de l'etat des taches */
212   }
213   _tache_c = 0;                      /* initialisation de la tache courante */
214   file_init();                       /* initialisation de la file           */
215
216   _tos = sp;                         /* Haut de la pile des t?ches */
217   _set_arm_mode_(ARMMODE_IRQ);       /* Passer en mode IRQ */
218   sp = _tos;                         /* sp_irq initial */
219   _set_arm_mode_(ARMMODE_SYS);       /* Repasser en mode SYS */
220
221   _irq_disable_();                   /* on interdit les interruptions */
222
223   /* Initialisation du timer ? 100 Hz */
224   tim1->tcmp = 10000;
225   tim1->tprer = 0;
226   tim1->tctl |= TCTL_TEN | TCTL_IRQEN | TCTL_CLKSOURCE_PERCLK16;
227
228   /* Initialisation de l'AITC */
229   aitc->intennum = TIMER1_INT;
230
231   active(cree(adr_tache));           /* creation et activation premiere tache */
232 }
233
234 /*-------------------------------------------------------------------------*
235  *                    --- Endormir la t?che courante ---                   *
236  * Entree : Neant                                                          *
237  * Sortie : Neant                                                          *
238  * Descrip: Endort la t?che courante et attribue le processeur ? la t?che  *
239  *          suivante.                                                      *
240  *                                                                         *
241  * Err. fatale:Neant                                                       *
242  *                                                                         *
243  *-------------------------------------------------------------------------*/
244
245 void dort(void) {
246
247 }
248
249 /*-------------------------------------------------------------------------*
250  *                    --- R?veille une t?che ---                           *
251  * Entree : num?ro de la t?che ? r?veiller                                 *
252  * Sortie : Neant                                                          *
253  * Descrip: R?veille une t?che. Les signaux de r?veil ne sont pas m?moris?s*
254  *                                                                         *
255  * Err. fatale:t?che non cr??e                                             *
256  *                                                                         *
```

```
257 *----------------------------------------------------------------------*/
258
259
260 void reveille(uint16_t t) {
261   CONTEXTE *p;
262
263 }
```

```
1   echo "hello world
```