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

Théophile DANCOISNE et Louis FRERET

Mai 2017

1 Ordonnanceur de tâches

Rappel : Le contexte d'un processus est correspond à une image des registres du processus à un instant t. En commutant la valeur du pointeur du registre du processeur, on effectue un changement de contexte.

```
/* NOYAUFILE.C */
/*-------------------------------------------------------------------------*
 *  gestion de la file d'attente des taches pretes et actives              *
 *  la file est rangee dans un tableau. ce fichier decrit toutes           *
 *  les primitives de base                                                 *
 *-------------------------------------------------------------------------*/

#include "serialio.h"
#include "noyau.h"

/* variables communes a toutes les procedures *
 *--------------------------------------------*/

static uint16_t _file[MAX_TACHES];    /* indice=numero de tache */
/* valeur=tache suivante  */
static uint16_t _queue;               /* valeur de la derniere tache */
/* pointe la prochaine tache a activer */

/*     initialisation de la file      *
 *------------------------------------*
entre  : sans
sortie : sans
description : la queue est initialisee vide, queue prend la valeur de tache
        impossible
*/

void file_init(void) {
  _queue = F_VIDE;
  for (int i = 0; i < MAX_TACHES; i++) {
    _file[i] = F_VIDE;
  }
}

/*        ajouter une tache dans la pile      *
 *--------------------------------------------*
entree : n numero de la tache a entrer
sortie : sans
description : ajoute la tache n en fin de pile
*/

void ajoute(uint16_t n) {
  if (_queue == F_VIDE) {
    _file[n] = n;
  }

  if (_file[n] == F_VIDE) {
    _file[n] = suivant();
```

```
48      _file[_queue] = n;
49    } else {
50      printf("Error: Tâche déjà existante.");
51    }
52
53  }
54
55  uint16_t predecesseur(uint16_t t);
56  /*           retire une tache de la file        *
57   *-----------------------------------------------*
58  entree : t numero de la tache a sortir
59  sortie : sans
60  description: sort la tache t de la file. L'ordre de la file n'est pas
61         modifie
62  */
63
64  void retire(uint16_t t) {
65    if (_file[t] == F_VIDE) {
66      printf("Error: Tâche inexistante.");
67    }
68
69    uint16_t pred_t = predecesseur(t);
70    _file[pred_t] = _file[t];
71
72    _file[t] = F_VIDE;
73  }
74
75  uint16_t predecesseur(uint16_t t) {
76    uint16_t pred_t;
77    for (int i = 0; i < MAX_TACHES; i++) {
78      if (_file[i] == t) {
79        pred_t = i;
80        break;
81      }
82    }
83    return pred_t;
84  }
85
86  /*        recherche du suivant a executer       *
87   *-----------------------------------------------*
88  entree : sans
89  sortie : t numero de la tache a activer
90  description : la tache a activer est sortie de la file. queue pointe la
91          suivante
92  */
93  uint16_t suivant(void) {
94    if (_queue == F_VIDE) {
95      printf("Error: Aucune tâche.");
96      return F_VIDE;
97    } else {
98      return _file[_queue];
99    }
100 }
101
102 /*     affichage du dernier element     *
103  *--------------------------------------*
104 entree : sans
105 sortie : sans
106 description : affiche la valeur de queue
107 */
108
109 void affic_queue(void) {
110   printf("Affichage de la queue:\n");
111
112   char *format = "Queue:\t%d\tValeur:\t%d\n";
113   printf(format, _queue, _file[_queue]);
114 }
115
```

```
/*     affichage de la file     *
 *-------------------------------*
entree : sans
sortie : sans
description : affiche les valeurs de la file
*/

void affic_file(void) {
  printf("Affichage de la file:\n");

  char *format = "Indice:\t%d\tValeur:\t%d\n";
  for (int i = 0; i < MAX_TACHES; i++) {
    printf(format, i, _file[i]);
  }
}
```

Listing 1 – noyaufil.c

L'ordonnancement ainsi implémenté est le plus simple du monde : chaque tâche est exécuté tour à tour sans priorité en respectant l'ordre défini par l'utilisateur.

```
// Ce programme a pour but de tester les différentes fonctions de NOYAUFIL.C
// de la partie 1 de "Réalisation dun mini noyau temps réel ARM"

#include <noyau.h>
#include <serialio.h>

int main(int argc, char **argv) {
  file_init();

  ajoute(3);
  ajoute(5);
  ajoute(1);
  ajoute(0);
  ajoute(2);

  affic_file();
  affic_queue();

  uint16_t tache_suivante = suivant();
  printf("Tâche suivante:\t%d\n", tache_suivante);

  affic_file();
  affic_queue();

  retire(0);

  affic_file();
  affic_queue();

  ajoute(6);

  affic_file();
  affic_queue();

  return 0;
}
```

Listing 2 – testfile.c

2 Gestion et commutation de tâches

```
void noyau_exit(void) {
  int j;
  _irq_disable_();                /* D?sactiver les interruptions */
  printf("Sortie du noyau\n");
```

```
5   for (j = 0; j < MAX_TACHES; j++)
6     printf("\nActivations tache %d : %d", j, compteurs[j]);
7   for (;;);                          /* Terminer l'ex?cution */
8 }
```

Listing 3 – noyau.c

```
1 void fin_tache(void) {
2   /* on interdit les interruptions */
3   _irq_disable_();
4   /* la tache est enlevee de la file des taches */
5   _contexte[_tache_c].status = CREE;
6   retire(_tache_c);
7   schedule();
8 }
```

Listing 4 – noyau.c

Listing 5 – noyau.c

Listing 6 – noyau.c

Listing 7 – noyau.c