NOM : PRENOM : N° Place :

Examen Final de MI11 - Printemps 2016

Durée de l'examen : 2 heures - Documents autorisés
Bien lire le texte. Longueur ne rime pas avec difficulté
Il n'y a pas de cloisonnement de connaissances entre les différentes parties du cours.
La clarté de la rédaction sera prise en compte ; n'oubliez pas de justifier vos réponses.

Attention aux changements de copies

Durée exercice : moins de 20 minutes

1. Linux embarqué ***(copie N°1)***

Question 1. (0.5 point) Afin de développer une application embarquée (pour architecture ARM par exemple), donnez les avantages à utiliser une chaîne de compilation croisée plutôt qu'une chaîne de compilation.

Question 2. (0.5 point) Donnez les avantages et les inconvénients d'un amorçage d'une cible par le réseau et sur mémoire flash. Dans quels cas utilise-t-on l'un ou l'autre ?

Question 3. Question 3. (0.5 point) Pourquoi cherche-t-on à minimiser la taille d'une distribution embarquée ? Quels sont les différents moyens pour y arriver ?

Question 4. (0.5 point) Même question pour le noyau.

Question 5. Question 5. (1 point) Décrivez une séquence complète de boot d'une cible par le réseau (similaire à celle utilisée en TP par exemple) ; de l'allumage jusqu'au login. Expliquez comment doit être configuré le serveur et quels fichiers doivent s'y trouver.

Durée exercice : moins de 20 minutes

2. Systèmes temps réel ***(copie N°2)***

Question 1. (0.5 point) Relier le concept de processus à des éléments concrets qu'il est possible d'identifier dans le code d'un exécutif temps réel.

Question 2. (0.5 point) Sachant une gigue de latence ε de 30 μs, une dérive d'horloge ρ de 2.10^{-5} s par seconde et une période de resynchronisation R_{int} de 0.5 s, quelle précision Π peut être atteinte par l'algorithme FTA dans un système contenant 25 horloges avec au maximum 3 horloges malicieuses ?

Question 3. (0.5 point) Dans un système ou toutes les tâches sont périodiques, chaque tâche consomme en moyenne 20% de la capacité CPU sur 1 seconde. On a quatre tâches en exécution. Le noyau est déclenché toutes les 20 ms. Un appel noyau consomme 0.2% de la capacité processeur en charge d'administration. Les événements apériodiques se signalent forcément par une interruption. Chaque évènement consomme 0.5% de capacité processeur pour son traitement propre et fait un appel noyau systématique. Sans se préoccuper de l'ordonnançabilité, combien d'évènements asynchrones pourra-t-on prendre en compte au maximum sur 1 seconde ?

Question 4. (0.5 point) Définir la notion de Ground State. Quelle est son utilité ?

Question 5. (1 point)

1 - Quand est-il nécessaire de recourir à une estimation d'état pour construire une image RT ?
2 - Définir le concept de composabilité.
3 - Quelle démarche faut-il adopter pour favoriser la composabilité, dans la construction d'un estimateur d'état qui utilise une observation produite sur un nœud différent de celui qui utilise l'image RT produite par cet estimateur ?

NOM : PRENOM : N° Place :

Durée exercice : 30 minutes

3. Mini-noyau temps réel préemptif ***(copie N°2)***

Dans vos réponses, vous pouvez utiliser du pseudo code pour écrire vos programmes !

A. Sauvegarde de contexte (3 points)

Dans le mini noyau du TP sur Arm, on a choisi d'attribuer une pile IRQ à chaque tâche. Le contexte de la tâche interrompue est sauvegardé sur sa propre pile IRQ.
Ceci n'est pas une obligation. On décide de changer le cahier des charges pour imposer une pile IRQ unique.

Question 1. Définir ce que représente le « contexte » de la tâche.

Question 2. Expliquer comment se déroule une sauvegarde/restitution de contexte dans la version du mini noyau développée en TP.

Question 3. Quelle autre solution qu'une pile IRQ spécifique à chaque tâche s'offre à vous pour sauvegarder le contexte des tâches ? Indiquez les modifications qu'il faudrait effectuer sur le code de `scheduler` et les définitions de « noyau.h » (ANNEXE 1). ***Le correcteur n'attend pas du code assembleur***, mais suffisamment de précisions sur la succession des opérations à effectuer tenant compte du fonctionnement du processeur ARM.

B. File vide (2 points)

Question 1. Dans le noyau ARM du TP, donnez un exemple de phénomène pouvant conduire à une file vide de tâches à ordonnancer.

En vous basant sur l'extrait du code donné en ANNEXE 1, expliquer ce qui se passe dans ce cas et les conséquences qui en découlent.

Question 2. Que feriez-vous pour supprimer ce problème et permettre au noyau de reprendre l'exécution de tâches ?

NOM : PRENOM : N° Place :

Durée exercice : moins de 20 minutes

4. Image mémoire (3 points) ***(copie N°3)***

On suppose qu'une plateforme embarquée ARM munie d'une mémoire Flash de type NOR d'une capacité de 1 Mo et d'une mémoire SDRAM d'une capacité de16 Mo. Cette plateforme est fournie avec une chaîne de compilation croisée GNU. Le script d'édition de liens associé est donné ci-dessous.

```
OUTPUT_FORMAT("elf-littlearm", "elf-bigarm",
              "elf-littlearm")
OUTPUT_ARCH(arm)
ENTRY(_start)

MEMORY {
   ROM (rx) : ORIGIN = 0x0, LENGTH = 1M
   RAM (rwx): ORIGIN = 0x02000000, LENGTH = 16M
}

SECTIONS {
   .text : {
      *(.vectors)
      *(.start)
      *(.text)
      . = ALIGN(4) ;
   } >ROM

   .fastcode : {
      __fastcode_load = LOADADDR(.fastcode) ;
      __fastcode_start = . ;
      *(.text.fastcode)
      . = ALIGN(4) ;
      __fastcode_end = . ;
   } >RAM AT>ROM

   .data : {
      __data_load = LOADADDR(.data) ;
      __data_start = . ;
      *(.data)
      . = ALIGN(4);
      __data_end = . ;
   } >RAM AT>ROM

   .rodata : {
      *(.rodata)
      . = ALIGN(4) ;
   } >ROM

   .bss : {
      __bss_start = . ;
      *(.bss)
      *(COMMON)
      . = ALIGN(4) ;
      __bss_end = . ;
   } >RAM

   .stack : {
      __stack_bottom = . ;
      . = ORIGIN(RAM) + LENGTH(RAM) ;
      __stack_top = . ;
   } >RAM
}
```

Question 1. Donnez graphiquement la structure de la carte mémoire de la cible pendant l'exécution d'un programme compilé avec cette chaîne de compilation croisée. **On suppose qu'aucune section d'entrée n'est vide**. Vous indiquerez la position de chaque section, les emplacements repérés par chacun des symboles définis dans le script et leur valeur numérique hexadécimale s'il est possible de la déterminer. Si des sections ont une adresse de chargement et une adresse virtuelle différentes, vous indiquerez les deux sur votre schéma.

Question 2. Décrivez les opérations successives que devrait réaliser le code de démarrage pour qu'une fonction C placée dans la section `.text.fastcode` s'exécute correctement. Dans quelle section devrait se trouver ce code de démarrage ?

Question 3. En supposant que la cible ne soit pas pourvue de mémoire cache, quel est l'intérêt de placer une fonction dans la section `.text.fastcode` ? Justifiez.

NOM : PRENOM : N° Place :

Durée exercice : 35 minutes

5. Ordonnancement de tâches avec relations de précédence (6 points) ***(copie N°4)***

Vous devez étudier l'ordonnançabilité de la configuration de tâches périodiques du Tableau 1, par la méthode d'ordonnancement EDF.

Tâche	r_i	C_i	d_i	T_i
τ_1	0	1	10	12
τ_2	0	2	9	12
τ_3	0	3	9	12
τ_4	0	1	10	12
τ_5	0	2	12	12

Tableau 1

Question 1. Déterminer le facteur d'utilisation du processeur et conclure sur l'ordonnançabilité de cette configuration dans le cas général de tâches indépendantes. Définir la période d'étude. Donner le chronogramme d'exécution des tâches.

Question 2. Les contraintes de précédence (Figure 1), sont rajoutées aux paramètres temporels de la configuration de tâches du tableau 1. Rappeler le principe de modification des paramètres des tâches proposé pour la prise en compte de ces contraintes. Calculer les nouveaux paramètres et donner le chronogramme d'exécution des tâches.

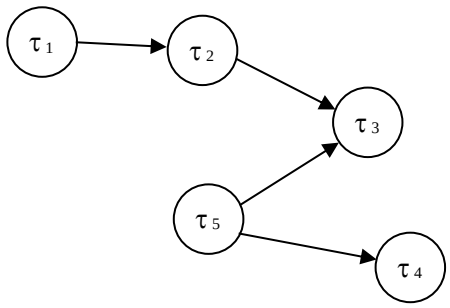

Figure 1. Contraintes de précédence

NOM : PRENOM : N° Place :

ANNEXE 1. **Extrait du code du micro noyau ARM**

Listing 1. noyau.h (extraits)

```
46  /*-----------------------------------------------------*
47  *mini  noyau  temps  reel  fonctionnant  sur  MC9328MXL
    *
48   *                    NOYAU.H
    *
49
    ****************************************************
50   * On definit dans ce fichier toutes les constantes et
    les structures       *
51   * necessaires au fonctionnement du noyau
52   *-----------------------------------------------------*/
53
54  ...
55
56  /* Les constantes */
57  /******************/
58
59  #define PILE_TACHE  4096             /* Taille maxi de
    la pile d'une tâche   */
60  #define PILE_IRQ    2048             /* Taille maxi de
    la pile IRQ par tâche */
61  #define MAX_TACHES  8                /* Nombre maximum
    de taches     */
62  #define F_VIDE      MAX_TACHES      /* numero de tache
    hors tableau */
63
64  /*  Definitions des fonctions dependant du materiel sous
    forme de code inline.
65  ****************************************************/
66  ...
67
68  #define _set_arm_mode_(mode) \
69    __asm__ __volatile__(\
70        "mrs  r0, cpsr\t\n"\
71        "bic  r0, r0, #0x1f\t\n"\
72        "orr  r0, r0, %0\t\n"\
73        "msr  cpsr_c, r0\t\n"\
74        "nop\t\n"\
75        :\
76        : "I" (mode)\
77        : "r0")
78
79  #define ARMMODE_USR   0x10
80  #define ARMMODE_SYS   0x1f
81  #define ARMMODE_SVC   0x13
82  #define ARMMODE_ABT   0x17
83  #define ARMMODE_UND   0x1b
84  #define ARMMODE_IRQ   0x12
85  #define ARMMODE_FIQ   0x11
86
87  /* Etat des taches */
88  /*******************/
89
90  #define NCREE   0         /* Etat non cree          */
91  #define CREE    0x8000    /* Etat cree ou dormant   */
92  #define PRET    0x9000    /* Etat eligible          */
93  #define SUSP    0xA000    /* Etat suspendu          */
94  #define EXEC    0xC000    /* Etat execution         */
95
96  /* definition des types */
97  /************************/
98
99  #define TACHE   void
100 typedef  TACHE    (*TACHE_ADR)  (void); /* pointeur de
    taches             */
101
```

```
1   /* definition du contexte d'interruption d'une
    tache */
2   /*****************************************/
3
4   /* Structure du contexte sur la pile IRQ */
5   typedef struct {
6     uint32_t r0;
7     uint32_t r1;
8     uint32_t r2;
9     uint32_t r3;
10    uint32_t r4;
11    uint32_t r5;
12    uint32_t r6;
13    uint32_t r7;
14    uint32_t r8;
15    uint32_t r9;
16    uint32_t r10;
17    uint32_t r11;
18    uint32_t r12;
19    uint32_t sp;
20    uint32_t lr;
21    uint32_t lr_irq;
22    uint32_t spsr_irq;
23  } REGS;
24
25  /* definition du contexte d'une tache */
26  /**************************************/
27
28  typedef struct {
29    TACHE_ADR tache_adr;       /* adresse de debut
    de la tache */
30    uint16_t  status;          /* etat courant de
    la tache     */
31    uint32_t  sp_ini;          /* valeur initiale
    de SP        */
32    uint32_t  sp_irq;          /* valeur courante
    de SP        */
33  } CONTEXTE;
34
35  /* Variables du noyau */
36  /**********************/
37
38   CONTEXTE _contexte[MAX_TACHES];  /* tableau des
    contextes  */
39   volatile uint16_t _tache_c;  /* numero de tache
    courante */
40   uint32_t   _tos; /* adresse du sommet de pile*/
41
42  /* Prototype des fonctions */
43  /***************************/
44
45  ...
```

NOM : PRENOM : N° Place :

Listing 2. noyau.c (extraits)

```
78  /*-------------------------------------------------------------------------*
79  *                        ORDONNANCEUR preemptif optimisé
80  *-------------------------------------------------------------------------*/
81  void __attribute__((naked)) scheduler( void )
82  {
83    register CONTEXTE *p;
84    register unsigned int sp asm("sp");  /* Pointeur de pile */
85
86    /* Sauvegarder le contexte complet sur la pile IRQ */
87    __asm__ volatile__(
88        "[redacted]"  /* Sauvegarde registres mode system */
89        "[redacted]"                    /* Attendre un cycle */
90        "[redacted]"    /* Ajustement pointeur de pile */
91        "[redacted]"                    /* Sauvegarde de spsr_irq */
92        "[redacted]");/* et de lr_irq */
93
94    if (_ack_timer)                 /* Réinitialiser le timer si nécessaire */
95    {
96      [redacted];
97      [redacted];
98    }
99    else
100   {
101     _ack_timer = 1;
102   }
103
104   _contexte[_tache_c].sp_irq = sp;/* memoriser le pointeur de pile */
105   _tache_c = suivant();           /* recherche du suivant */
106   if (_tache_c == F_VIDE)
107   {
108     printf("Plus rien à ordonnancer.\n");
109     noyau_exit();                 /* Sortie du noyau */
110   }
111   compteurs[_tache_c]++;          /* Incrémenter le compteur d'activations  */
112   p = &_contexte[_tache_c];       /* p pointe sur la nouvelle tache courante*/
113
114   if (p->status == PRET)          /* tache prete ? */
115   {
116     sp = p->sp_ini;               /* Charger sp_irq initial */
117     _set_arm_mode_(ARMMODE_SYS);  /* Passer en mode système */
118     sp = p->sp_ini - PILE_IRQ;    /* Charger sp_sys initial */
119     p->status = EXEC;             /* status tache -> execution */
120     _irq_enable_();               /* autoriser les interuptions   */
121     (*p->tache_adr)();            /* lancement de la tâche */
122   }
123   else
124   {
125     sp = p->sp_irq;               /* tache deja en execution, restaurer sp_irq */
126   }
127
128   /* Restaurer le contexte complet depuis la pile IRQ */
129   __asm__ volatile__(
130       "[redacted]"  /* Restaurer lr_irq */
131       "[redacted]"                  /* et spsr_irq */
132       "[redacted]"  /* Restaurer registres mode system */
133       "[redacted]"                  /* Attendre un cycle */
134       "[redacted]"    /* Ajuster pointeur de pile irq */
135       "[redacted]");   /* Retour d'exception */
136 }
```