M2 Examen Informatique Embarquée

16 Décembre 2011 9H00-12H00 - Documents autorisés[1].

Total sur *90 points*. Questions indépendantes les unes des autres.
Regardez le barème avant de vous lancer.

1. **Système Embarqué** *(6 points)*

   Donnez une définition de ce qu'est un système embarqué. Parmi les contraintes qui s'appliquaient historiquement aux systèmes embarqués, quelles sont celles qui ont tendance à s'amoindrir et à faire que les systèmes embarqués se distinguent (un peu) moins des systèmes informatiques « conventionnels » ?

   - Donnez un exemple de système embarqué dans lequel la consommation énergétique doit être prise en compte sérieusement.
   - Donnez un exemple de système embarqué dans lequel le poids (en kg) du système est important.

2. **Langage C** *(20 points)*

   2.1. **Sizeof.** *(4 points)* Sur une machine dont la taille des mots est de 32 bits, quelle sera la taille (telle que retournée par « sizeof ») des structures suivantes ?

```
struct s1 {
    char       c1[3];
    long long  k;
    char       c2;
    char      *pt;
    char       c3;
};
```

```
struct s2{
    int        i;
    char       c1;
    long long  k;
    char       c2;
} __attribute__((__packed__));
```

   2.2. **Volatile.** *(3 points)* Quelle est l'utilité du mot clé `volatile` en C?

   2.3. **Impression** *(3 points)* Que va afficher la séquence ci-dessous ? Pourquoi ?

```
char  *str = "abc";
printf ("%c %c \n", *str++, *str++);
```

   2.4. **Liste circulaire doublement chaînée** *(10 points)*

   Définissez une structure (`struct dlist`) pour manipuler une liste circulaire doublement chaînée. Définissez et implémentez

   - une fonction d'initialisation d'une telle structure (`dlist_init`)
   - une fonction pour ajouter un élément dans une liste existante (`dlist_add`)
   - une fonction pour enlever un élément d'une liste existante (`dlist_del`)

1 Sauf documents électroniques

3. Synchronisation / Ordonnancement *(15 points)*

3.1. Deux Sémaphores *(6 points)*

Dans l'**extrait de code** ci-dessous, on suppose que deux threads sont créées. L'une exécute la fonction threadA et l'autre threadB. Elles sont toutes deux en classe d'ordonnancement FIFO au même niveau de priorité sur une machine mono-processeur. On suppose que threadA est la première thread active. On suppose aussi que la thread principale (main) attend la fin d'exécution des 2 threads précédemment créées.

Quel est l'ordonnancement résultant entre ces 2 threads : vous commencerez par définir les états d'une thread du point de vue de l'ordonnanceur et vous indiquerez quelles transitions suivent les 2 threads dans le scénario ci-dessous. Apporteriez-vous une modification à ce programme, pourquoi ?

```
void * threadA (void * arg) { // Extrait de code
    for(i = 0; i < LOOP; i++) {
        sem_post(&s2);
        sem_wait(&s1);
    }
}
void * threadB (void * arg) {// Extrait de code
    for(i = 0; i < LOOP; i++) {
        sem_post(&s1);
        sem_wait(&s2);
    }
}
main() {// Extrait de code
    sem_init(&s1, 0, 1);
    sem_init(&s2, 0, 0);
    pthread_create(&thA, &attr, threadA, NULL);
    pthread_create(&thB, &attr, threadB, NULL);
}
```

3.2. Un Sémaphore *(4 points)*

```
void * threadA (void * arg) {// Extrait de code
    ...
    for(i = 0; i < LOOP; i++) {
        sem_post(&s1);
        sem_wait(&s1);
    }
}
void * threadB (void * arg) {// Extrait de code
    ...
    for(i = 0; i < LOOP; i++) {
        sem_wait(&s1);
        sem_post(&s1);
    }
}
```

Suivant les mêmes hypothèses que précédemment, quel est l'ordonnancement obtenu lors de l'exécution du programme, si le code des threads est remplacé par le code ci_dessus ?

3.3. **Verrous** *(5 points)*

Sur un système n'ayant pas de mécanisme de détection de dépendance circulaire, on a 3 threads T1, T2 et T3 soumises à un ordonnancement de type FIFO avec la relation suivante entre leurs priorités : Prio(T1) < Prio(T2) < Prio(T3) et 3 mutex m1, m2 et m3 correctement initialisés sans attribut particulier et pas encore utilisés. L'exécution se déroule comme suit :

3.3.a) T1 exécute l'appel : `pthread_mutex_lock(&m1);`
3.3.b) T2 préempte T1
3.3.c) T2 exécute l'appel : `pthread_mutex_lock(&m2);`
3.3.d) T3 préempte T2
3.3.e) T3 invoque la fonction : `pthread_mutex_lock(&m3);`
3.3.f) T3 invoque la fonction : `pthread_mutex_lock(&m1);`
3.3.g) T1 invoque la fonction : `pthread_mutex_lock(&m2);`
3.3.h) T2 invoque la fonction : `pthread_mutex_lock(&m3);`

3.3.1 Dans quelle situation se trouve-t-on à la fin de l 'étape 3.3.h)? Soyez clair et précis. Comment éviteriez-vous ce genre de situation ? Que se passerait-il si au lieu d'être soumises à un ordonnancement de type FIFO, ces threads étaient soumises à un ordonnancement de type temps-partagé?
3.3.2 Si l'étape 3.3.h) était remplacée par un appel : `thread_mutex_unlock(&m2)`, que se passerait-il ?
3.3.3 Si les mutex m1, m2 et m3 supportaient l'héritage de priorité, la situation en 3.3.h)serait-elle différente ?

4. **Mémoires Flash** *(8 points)*

4.1. Quels sont les 2 types de mémoires Flash ? Quelles différences y a-t-il entre ces 2 types de mémoires ? Quel type est principalement utilisé dans les clés USB ? Est-il possible d'exécuter un programme stocké dans une mémoire Flash sans recopier son code (respectivement ses données) vers la RAM ? *(6 points)*

4.2. Quel est le principal problème posé par les mémoires Flash ? *(2 points)*

5. **OS, Micro-noyaux et OS embarqués** *(9 points)*

5.1. Quels principes président à la définition de l'architecture des micro-noyaux comme L4, par opposition aux systèmes monolithiques (comme Linux) ? *(2 points)*

5.2. Quels services sont typiquement offerts par un micro-noyau ? *(4 points)*

5.3. Comparez les types de services rendus par le système eCos (sans prendre en compte ses librairies) et ceux rendus par L4. *(3 points)*

6. **QEMU / Linux** *(10 points)*

6.1. Quelles sont les principales étapes de la construction d'un noyau Linux à partir des sources ?*(2 points)*

6.2. Dans le TP N°5, pourquoi doit-on exécuter la commande genimage (ou sa variante) à chaque fois que l'on modifie le (contenu du) répertoire « root » ? *(2 points)*

6.3. Dans le TP N°5, lors de l'exécution de la commande suivante (simplifiée ici) sur une machine Linux Debian de la salle de TP,

```
qemu-system-x86_64 -kernel bzImage -hda tst.img -append 'root=/dev/hda1'
```

6.3.a) Comment est spécifié le noyau Linux que l'on veut exécuter ? *(1 point)*

6.3.b) Sur quelle machine va-t-il s'exécuter ? *(1 point)*

6.3.c) Quel est le processus (nom de commande) qui sera visible par ps sur la machine Debian de la salle de TP ? *(1 point)*

6.3.d) Pour quelles raisons, le système Linux sur QEMU peut-il dire « init not found » ? *(3pts)*

7. **MMU** *(12 points)*

7.1. *(6 points)* Qu'est-ce qu'une MMU ? Quel est son rôle ? Est-ce un composant matériel ou logiciel ? Linux peut-il fonctionner une machine sans MMU ? Pourquoi ? eCos peut-il tourner une machine sans (respectivement avec) MMU ? Pourquoi ?

7.2. *(6 points)* Sur un système avec MMU (respectivement sans MMU), peut-on :

- Exécuter un programme dont la taille dépasse la taille de la mémoire physique ?
- Exécuter simultanément plusieurs programmes dont la taille cumulée dépasse la taille de la mémoire physique ?
- Exécuter simultanément plusieurs copies d'un même programme ELF ?

Vous ne vous contenterez pas de répondre par oui/non aux questions ci-dessus, mais vous étayerez ces réponses.

8. **Compilation, tailles de programmes.** *(10 points)*

8.1. **Tailles.** *(8 points)* On a procédé à la génération de « BusyBox » de deux manières :

- une première fois, en choisissant une édition de liens statique (binaire busybox.static)
- une deuxième fois, en choisissant une édition de liens dynamique (binaire busybox).

On a appliqué la commande « size » à ces deux versions, ainsi qu'à la bibliothèque dynamique C (on supposera que c'est la seule utilisée). Voilà le résultat de ces différentes commandes:

```
# size busybox.static
   text       data        bss        dec        hex    filename
1904387       4280      26512    1935179     1d874b    busybox.static
# size busybox
   text       data        bss        dec        hex    filename
 855597       4268      10064     869929      d4629    busybox
# size /lib/libc.so.6
   text       data        bss        dec        hex    filename
1405854      17976      19168    1442998     1604b6    /lib/libc.so.6
```

8.1.a) Si on utilise la version statique de BusyBox, quelle sera la taille utilisée en mémoire (sur une machine à base de processeurs Intel) par l'exécution simultanée des 2 commandes suivantes : ls, et cp?

8.1.b) Si on utilise la version dynamique de BusyBox, quelle sera la taille utilisée en mémoire (sur une machine à base de processeurs Intel) par l'exécution simultanée des 2 mêmes commandes : ls, et cp?

Dans un cas comme dans l'autre, on supposera que l'intégralité du code et des données sont chargées en mémoire et on fera les calculs en arrondissant les tailles en fonction de la taille des pages de la machine (4K octets). On ne tiendra pas compte des allocations dynamiques (tas), ni de la pile. Ces commandes sont supposées être mono-thread. Vous motiverez vos calculs en expliquant comment vous procédez et pourquoi vous procédez de cette manière.

8.2. **Pourquoi** utilise-t-on BusyBox dans des systèmes embarqués? *(2 points)*