SYSTEMES INFORMATIQUES ET APPLICATIONS CONCURRENTES UE NFP137

EXAMEN 1ère Session 2008

Date : dimanche 22 juin 2008
Durée : 2 heures
Heure : 9h à 11h
Lieu : salle 11-A3-33

TOUS DOCUMENTS PAPIERS AUTORISES
(les ordinateurs portables ne sont pas autorisés)

Barême indicatif
A Gestion de mémoire : 8 points
B Communication et Synchronisation de processus : 12 points
Question 1 : 2 points
Question 2 : 4 points
Question 3 : 6 points

Ne pas écrire au crayon ni à l'encre rouge
(sous peine de nullité, selon le règlement des examens)

A Étude d’une gestion de mémoire

L'architecture matérielle d'un système embarqué est basée sur un CPU comprenant 2 modes d'exécution, superviseur et utilisateur, représentés dans le registre d'état du CPU.

- **le mode superviseur** permet l'accès aux ressources protégées et l'exécution des instructions privilégiées du CPU. C'est le mode de démarrage du CPU.

- **le mode utilisateur** interdit l'accès aux ressources protégées et l'exécution des instructions privilégiées.

Le CPU inclut une MPU (Memory Protection Unit) permettant de délimiter 8 régions de mémoire indépendantes.

Chaque région de la MPU est définie par les 4 informations suivantes :

- le bit de validité de la région,
- l'adresse sur 24 bits de début de la région,
- la taille de la région en multiple de 256 octets,
- les droits d'accès à la région : R (read), W (write), X (execute).

La MPU est systématiquement court-circuitée lorsque le CPU est en mode superviseur. Inversement, la MPU est toujours mise en oeuvre lorsque le CPU est en mode utilisateur.

Seules les régions valides, c'est-à-dire celles dont le bit de validité est à 1, sont considérées par la MPU.

A chaque accès mémoire effectué par le CPU en mode utilisateur, la MPU compare l'adresse et le type d'accès aux informations contenues dans chacune des régions valides.

- Si une région inclut l'adresse accédée avec le type d'accès demandé, la MPU autorise l'accès en mémoire.
- Sinon, la MPU invalide l'accès et déclenche une exception spécifique du CPU réservée à cet usage.

Le registre d'état du CPU et les registres de la MPU sont des ressources protégées qui ne sont accessibles qu'en mode superviseur. Le passage en mode utilisateur est effectué par l'intermédiaire d'une instruction privilégiée du CPU.

Le programme binaire de chaque application est constitué de 3 parties :

- la zone du code contenant les instructions à exécuter
- la zone des données globales initialisées à une valeur particulière au démarrage du programme
- la zone des données globales initialisées à zéro au démarrage du programme (zone appellée "bss" par la suite)

Le système attribue une pile utilisateur de 2048 octets à [la thread de] chaque application.
Enfin, le système ne fournit pas de service d'allocation dynamique de mémoire aux applications.

L'ensemble des applications est statique (pas de chargement dynamique d'application) et connu par le système au moment de son démarrage.

Cet ensemble est composé des 4 applications suivantes : (taille code, taille données, taille bss en octets).

	Taille_zone_Code	**Taille_zone_Données**	**Taille Bss**
A1	1048	60	180
A2	1800	510	258
A3	3400	0	250
A4	8196	110	914

Question 1.
Indiquer le nombre MINIMUM de régions de la MPU nécessaires à chaque application, et à quoi chaque région est associée par le système

Question 2.

Déterminer la taille totale de mémoire que doit allouer le système pour chacune des applications A1, A2, A3 et A4.

Question 3.1

Indiquer comment répartir les régions de la MPU entre les applications A1, A2, A3 et A4.

Question 3.2

Préciser les informations de gestion mémoire que le système doit conserver dans le contexte de chaque application.

Question 4.1

Décrire les principales opérations effectuées par le système sur la MPU lorsqu'il change l'application courante.

B Communication et synchronisation de processus

Exercice 1 : rendez-vous entre processus

Dans ce qui suit, les processus sont identifiés par un numéro : P0 par 0, Pi par i …

On souhaite programmer un rendez-vous symétrique entre deux processus cycliques P0 et P1 . Le schéma des processus est le suivant :

```
Contexte commun
// déclaration et initialisation des sémaphores

 Processus P0                  Processus P1
 Début                         Début
  Tant que vrai faire           Tant que vrai faire
     Réveiller P1 ;                Réveiller P0 ;
     Attendre_réveil_0 ;           Attendre_réveil_1  ;
     Code0 ;                       Code1 ;
  Fin ;                         Fin ;
```

Question 1.

Programmer les éléments **Réveiller P0, Réveiller P1, Attendre_réveil_0 , Attendre_réveil_1** *en utilisant des sémaphores et les primitives P, V, E0. Ne pas oublier de déclarer et d'initialiser les sémaphores.*

On souhaite maintenant programmer un rendez-vous entre N processus parallèles.
Une idée est de compter le nombre de processus arrivant au rendez-vous ; un processus est bloqué s'il n'est pas le dernier, s'il est le dernier processus il autorise tous les processus à franchir le rendez-vous. On ne fait pas d'hypothèse sur l'ordre de libération des processus.
Les processus sont identifiés par un numéro entier de 0 à N-1.

Schéma de programmation des processus :

```
Contexte commun
entier compte =0 ;
//déclaration et initialisation des sémaphores à programmer
 Processus Pi
Début
  Tant que vrai faire
   compte= compte + 1 ;
   Si compte = N alors
    compte=0 ;
    // autoriser les N processus à programmer
   finsi ;
   //blocage si le processus n'est pas le dernier à
programmer
  Code_i ;// reste du programme de Pi
 Fait ;
Fin ;
```

Question 2.

Compléter la programmation du schéma d'un processus Pi en utilisant des sémaphores et les primitives P, V et E0.

Plusieurs solutions sont possibles pour le blocage et le déblocage des processus : on pourra utiliser des sémaphores privés ou un seul sémaphore pour tous les processus. On notera que `compte` est une variable partagée et qu'il faut assurer la cohérence de sa valeur.

Exercice 2 : diffusion de messages via un tampon de mémoire

Un processus Emetteur diffuse des messages à trois processus Recepteur en déposant ces messages dans un tampon de mémoire partagé, contenant 10 cases.
Chaque Recepteur prélève tout message une fois et une seule.

Le dernier Recepteur à prélever un message donné libère la case correspondante du tampon.

```
Contexte commun
 message tampon[10] ;
 entier compte[10] ; pour i de 0 à 9 faire compte[i]=0 ;
 semaphore mutexcase[10] ; //accès cohérent au tampon
 semaphore casePleine[10] ;
//casepleine[i] autorisation des trois Recepteur à prélever un
//message dans la case i du tampon
semaphore Vide ;//autorisation de l'Emetteur à déposer un message
//initialisation des sémaphores à programmer

Processus Emetteur
entier queue=0 ;//indice de production
message m ;
tant que vrai faire
 produire(m) ;
 //autorisation de déposer un message à programmer
 tampon[queue]=m ;
 //ajout de 3 autorisations pour prelever dans la case
 // d'indice queue à programmer
 queue = (queue +1)%10 ;
fait ;

Processus UnRecepteur
entier tete=0 ; //indice de consommation pour ce Recepteur
message m ;
tant que vrai faire
 //autorisation de prélever dans la case d'indice tete à
programmer;
 //accès cohérent à la case d'indice tete à programmer;
 m=tampon[tete] ;
 compte[tete]= compteur[tete]+1 ;
 si compteur[tete]== 3 alors
  compte[tete]= 0 ;
  //libérer une case, ajout d'une autorisation de déposer à
programmer
 finsi ;
 //libérer l'accès cohérent à la case d'indice tete à
programmer
  tete=(tete+1)%10 ;
fait ;
```

Question 3.

En utilisant les sémaphores proposés et les primitives P, V et E0, compléter la programmation des processus Emetteur et UnRecepteur.

Corrigé indicatif

A Gestion de mémoire

Q1

Pour respecter les règles minimales de protection mémoire, il suffit d'attribuer
à chaque application deux régions de la MPU :
  - une région associée à la zone de code ayant les droits lecture/exécution
  - une région de données ayant les droits lecture/écriture et regroupant
    - la pile d'exécution de la thread de l'application
    - la zone des données globales initialisées
    - la zone des bss

Q2

La taille d'une région de la MPU étant un multiple de 256 octets, le système doit
allouer la mémoire des régions de chaque application selon la formule suivante :

  taille région code = ((taille code application + 256 - 1) / 256) * 256

  taille totale données = taille pile + taille données globales + taille bss
  taille région données = ((taille totale données + 256 - 1) / 256) * 256

Avec cette formule, la mémoire allouée à chaque application est :

	taille région code en octets	taille région données en octets	taille totale en octets
A1 :	1048	2304	3352
A2 :	2048	2816	4864
A3 :	3584	2304	5888

A4 : 8448 3072 11520

Q3

Le système peut attribuer 2 régions de la MPU à chaque application, et initialiser
chacune d'elle avec les informations suivantes conservées dans le contexte de
chaque application :
- adresse de la région de code
- taille de la région de code
- adresse de la région de données
- taille de la région de données

Q4

Lorsqu'il change d'application courante, le système doit uniquement :
1- invalider les 2 régions de la MPU attribuées à l'ancienne application,
en mettant à zéro leur bit de validité.
2- valider les 2 régions de la MPU attribuées à la nouvelle application,
en mettant à un leur bit de validité.

Les autres régions de la MPU n'ont pas à être modifiées, elles restent invalides.

B Communication et Synchronisation de processus

Exercice 1 : rendez-vous entre processus

Question 1.

Contexte commun
Semaphore spriv0, spriv1 ;
E0(spriv0,0) ; E0(spriv1,0) ;

```
Processus P0                Processus P1
Début                       Début
 Tant que vrai faire         Tant que vrai faire
    V(spriv1) ;                 V(spriv0) ;
    P(spriv0)                   P(spriv1) ;
    Code0 ;                     Code1 ;
 Fin ;                       Fin ;
```

Question 2.

```
 Avec sémaphores privés
Contexte commun
entier compte =0 ;
semaphore mutex ; E0(mutex,1) ;
semaphore spriv[N] ;
pour i de 0 à N-1 faire E0(spriv[i],0) ; fait ;
Processus Pi
Début
  Tant que vrai faire
   P(mutex) ;
   compte= compte + 1 ;
   Si compte = N alors
    compte=0 ;
   pour i de 0 à N-1 faire V(spriv[i]) ; fait ;
   finsi ;
   V(mutex) ;
   P(spriv[i]) ;//blocage si i n'est pas le dernier
   Code_i ;
  fait ;

Avec un seul semaphore de blocage :
Contexte commun
entier compte =0 ;
semaphore mutex ; E0(mutex,1) ;
semaphore rendez_vous ;
 E0(rendez_vous,0) ;
Processus Pi
Début
  Tant que vrai faire
   P(mutex) ;
   compte= compte + 1 ;
   si compte = N alors
    compte=0 ;
    pour i de 1 à N faire V(rendez-vous) ; fait ;
   finsi ;
```

```
   V(mutex) ;
   P(rendez_vous) ;//blocage si i n'est pas le dernier
   Code_i ;
  fait ;
```

Exercice 2 : diffusion de messages via un tampon de mémoire

Question 3.

```
Contexte commun
message tampon[10] ;
entier compte[10] ; pour i de 0 à 9 faire compte[i]=0 ;
//compte[i] nombre de Recepteur ayant prélevé le message de la case i
semaphore Mutexcase[10] ; //accès cohérent au tampon
semaphore CasePleine[10] ;
// casepleine[i]autorisation des trois Recepteur à prélever la case i
//du tampon
semaphore Vide ;//autorisation de l'Emetteur à déposer un message

E0(vide,10) ;
Pour i de 0 à 9 faire E0(CasePleine[i],0) ;
Pour i de 0 à 9 faire E0(Mutexcase[i],1) ;
Processus Emetteur
entier queue=0 ;//indice de production
message m ;
tant que vrai faire
 produire(m) ;
 P(vide) ;
 tampon[queue]=m ;
 pour i de 1 à 3 faire V(CasePleine[queue]) ; fait ;
 queue = (queue +1)%10 ;
fait ;
Processus UnRecepteur
entier tete=0 ; //indice de consommation pour ce Recepteur
message m ;
tant que vrai faire
 P(Casepleine[tete]) ;
 P(Mutexcase[tete]) ;
 m=tampon[tete] ;
 compte[tete]= compte[tete]+1 ;
 si compte[tete]== 3 alors
  compte[tete]= 0 ;
  V(vide) ;
 finsi ;
 V(Mutexcase[tete]) ;
tete=(tete+1)%10 ;
```

```
fait ;
```