SYSTÈMES ET RÉSEAUX INFORMATIQUES

COURS B4 : HTO(19339) et ICPJ(21937)
CYCLE PROBATOIRE INFORMATIQUE
(Conception et développement informatique)

EXAMEN DU 21 JUIN 2003

portant sur l'enseignement des SYSTÈMES INFORMATIQUES

Date : samedi 21 juin 2003
Heure : 09h à 12h

Lieu : ARCUEIL salle A4

TOUS DOCUMENTS PAPIERS AUTORISES

(les ordinateurs portables ne sont pas autorisés)

****

Texte de 6 pages

L'examen comprend :
des questions sur la gestion des ressources
des questions sur la synchronisation

Chaque question peut-être traitée indépendamment.

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

****

Barème indicatif :

A. COMPOSANTS AVEC COOPÉRATION DE PROCESSUS : 5 points

B. SERVEUR DE SIGNATURES D'IMAGES : 4 points

C. SERVEUR AVEC TROIS DISQUES : 5 points

D. ACCÈS DISQUES : 6 points

A. COMPOSANTS AVEC COOPÉRATION DE PROCESSUS

Une société de certification de logiciel a trouvé des formes inhabituelles de programmation du paradigme producteur consommateur (dans le cas où plusieurs producteurs et plusieurs consommateurs se partagent le même tampon). Elle vous demande votre expertise.

QUESTION A. Pour chacun des cas suivants, vous devez dire si le paradigme producteur-consommateur est correctement programmé (inhabituel, mais correct). Si ce n'est pas le cas, vous donnerez un exemple de dysfonctionnement, et corrigerez le programme pour qu'il devienne correct.

Tous les cas utilisent le contexte commun suivant :

```
N : Constant Integer := ...;
type Tampon is array(0..N-1) of Message;
T : Tampon;
Tete, Queue: Mod N := 0;
NbMess  : Integer := 0; -- intervient pour le 4e cas seulement
Mutex : Semaphore; E0(Mutex, 1);
Nplein, Nvide : Semaphore;
E0(Nplein, 0); E0(Nvide, N);
task type Producteur;
task type Consommateur;
```

Premier cas

```
task body Producteur is
   X : Message;
begin
 loop
  Produire (X);
  P(Nvide);
  P(Mutex);
  T(Queue) := X;
  Queue:= Queue + 1; -- + modulo N
  V(Mutex);
  V(Nplein);
 end loop;
end Producteur;
```

```
task body Consommateur is
   Y : Message;
begin
 loop
   P(Nplein);
   P(Mutex);
   Tete := Tete + 1; -- + modulo N
   Y := T(Tete);
   V(Mutex);
   V(Nvide);
   Consommer(Y);
 end loop;
end Consommateur;
```

Deuxième cas

```
task body Producteur is
   X : Message;
begin
 loop
  Produire (X);
  P(Nvide);
  P(Mutex);
  Queue:= Queue + 1; -- + modulo N
  T(Queue) := X;
  V(Mutex);
  V(Nplein);
 end loop;
end Producteur;
```

```
task body Consommateur is
   Y : Message;
begin
 loop
   P(Nplein);
   P(Mutex);
   Tete := Tete + 1; -- + modulo N
   Y := T(Tete);
   V(Mutex);
   V(Nvide);
   Consommer(Y);
 end loop;
end Consommateur;
```

Troisième cas

```
task body Producteur is
   X : Message;
begin
 loop
  Produire (X);
  P(Mutex);
  P(Nvide);
  Queue:= Queue + 1; -- + modulo N
  T(Queue) := X;
  V(Mutex);
  V(Nplein);
 end loop;
end Producteur;
```

```
task body Consommateur is
   Y : Message;
begin
 loop
   P(Mutex);
   P(Nplein);
   Tete := Tete + 1; -- + modulo N
   Y := T(Tete);
   V(Mutex);
   V(Nvide);
   Consommer(Y);
 end loop;
end Consommateur;
```

Quatrième cas

```
task body Producteur is
   X : Message; Numero : Integer;
begin
 loop
  Produire (X);
  P(Nvide);
  P(Mutex);
  NbMess := NbMess + 1;
  Numero := NbMess;
  V(Mutex);
  T(Numero modulo N) := X;
  V(Nplein);
 end loop;
end Producteur;
```

```
task body Consommateur is
   Y : Message;
begin
 loop
   P(Nplein);
   P(Mutex);
   Tete := Tete + 1; -- + modulo N
   Y := T(Tete);

   V(Mutex);
   V(Nvide);
   Consommer(Y);
 end loop;
end Consommateur;
```

B. SERVEUR DE SIGNATURES D'IMAGES

Un serveur de signatures d'images reçoit des requêtes de la forme (source, traitement, puit) où :

- source est une adresse d'une image sur un disque d'entrée,
- traitement consiste à ajouter une signature subliminale sur l'image,
- puit est une adresse d'un deuxième disque où l'image est rangée après le traitement.

QUESTION B.

a) Rappeler brièvement le principe et l'intérêt de la multiprogrammation et ce qu'on appelle taux de multiprogrammation. Comme il y a beaucoup de temps d'attente d'entrée-sortie, on peut utilement envisager d'exécuter en multiprogrammation plusieurs instances du serveur. On suppose que les accès disques sont gérés par des dispositifs d'accès direct à la mémoire (DMA ou canaux d'entrée-sortie) et que l'architecture est monoprocesseur. La lecture de l'image "source" peut se faire entièrement avant le traitement et l'écriture de l'image "puit" peut se faire après le traitement. Si on suppose un temps moyen de lecture ou d'écriture de 60 ms pour une image et un temps de traitement moyen de 20 ms, quel serait le taux de multiprogrammation à adopter pour utiliser l'unité centrale à 100%.

b) Pour mettre en place la multiprogrammation, il faut allouer de la mémoire à chaque processus. On suppose que chaque processus travaille avec trois zones de un megaoctet : une zone pour l'image "source", une zone pour l'image "puit" et une zone pour le programme de traitement. Ces zones sont acquises dynamiquement par les processus. Chaque processus multiprogrammé commence par demander une zone pour son image "source", puis quand elle est chargée, il demande une deuxième zone pour son programme de traitement et ensuite il demande une zone pour l'image "puit". On veut mettre en place un taux de multiprogrammation de 6. Quel est le nombre minimal M de zones à prévoir pour cette allocation dynamique si on veut être sûr de ne pas avoir d'interblocage?

c) Programmer, avec des sémaphores, l'allocateur de zones. Utiliser le canevas suivant :

```
subtype IdZone is Integer range  1..M;
paquetage Zone is
      procedure Allouer(I: out IdZone);
      procedure Restituer(J : in IdZone);
end Zone;

package body Zone is
Disponible : array(1..M) of Boolean := (others => True); -- indique les zones non allouées
S_Z, Mutex : Semaphore;
procedure Allouer(I: out IdZone) is begin .. -- à programmer; end Allouer;
procedure Restituer(J : in IdZone) is begin .. -- à programmer; end restituer;
begin..--programmer les initialisations; end Zone;
```

C. SERVEUR AVEC TROIS DISQUES

Devant le nombre d'images à traiter, on ajoute un troisième disque et on banalise l'usage des trois disques. Chacun peut contenir des images "source" et des images "puit". Chaque disque n'est utilisé que par un processus à la fois et devient une ressource allouée dynamiquement.
Soit alors les paquetages suivants :

```
generic
package Disque is
      procedure Obtenir;
      procedure Rendre;
end Disque;

package Disque1 is new Disque;
package Disque2 is new Disque;
package Disque3 is new Disque;

package body Disque is
-- à programmer
end Disque;

package Controle  is
      procedure Assurer;
      procedure Relacher;
end Controle;

package body Controle  is
      procedure Assurer is begin null; end Assurer; -- ne fait rien
      procedure Relacher is begin null; end Relacher; -- ne fait rien
begin null; end Controle;
```

Le service comprend six processus serveurs et les requêtes reçues par le service sont alors aiguillées vers le serveur gérant (le ou) les disques utilisés.

QUESTION C.

a) Programmer le package body Disque (les procédures Obtenir et Rendre et l'initialisation) avec des sémaphores. Montrer qu'il peut y avoir interblocage si on programme alors les six serveurs comme suit :

(à ce stade, les appels à Controle peuvent être ignorés car ils correspondent à une action vide).

```
task Serveur1 is
begin loop;
  Controle.Assurer;
  Disque1.Obtenir;
  Lecture"source";
 -- utilise Disque1
  Traitement;
  Ecriture"puit";
  --utilise encore Disque1
  Disque1.Rendre;
  Controle.Relacher;
end loop;
end Serveur1 ;
```

```
task Serveur2 is
begin loop;
  Controle.Assurer;
  Disque2.Obtenir;
  Lecture"source";
 -- utilise Disque2
  Traitement;
  Ecriture"puit";
  --utilise encore Disque2
  Disque2.Rendre;
  Controle.Relacher;
end loop;
end Serveur2 ;
```

```
task Serveur3 is
begin loop;
  Controle.Assurer;
  Disque3.Obtenir;
  Lecture"source";
 -- utilise Disque3
  Traitement;
  Ecriture"puit";
  --utilise encore Disque3
  Disque3.Rendre;
  Controle.Relacher;
end loop;
end Serveur3 ;
```

```
task Serveur12 is
begin loop;
  Controle.Assurer;
  Disque1.Obtenir;
  Disque2.Obtenir;
  Lecture"source";
 -- utilise Disque1
  Disque1.Rendre;
  Traitement;
  Ecriture"puit";
  --utilise Disque2
  Disque2.Rendre;
  Controle.Relacher;
end loop;
end Serveur12 ;
```

```
task Serveur23 is
begin loop;
  Controle.Assurer;
  Disque2.Obtenir;
  Disque3.Obtenir;
  Lecture"source";
 -- utilise Disque2
  Disque2.Rendre;
  Traitement;
  Ecriture"puit";
  --utilise Disque3
  Disque3.Rendre;
  Controle.Relacher;
end loop;
end Serveur23 ;
```

```
task Serveur31 is
begin loop;
  Controle.Assurer;
  Disque3.Obtenir;
  Disque1.Obtenir;
  Lecture"source";
 -- utilise Disque3
  Disque3.Rendre;
  Traitement;
  Ecriture"puit";
  --utilise Disque1
  Disque1.Rendre;
  Controle.Relacher;
end loop;
end Serveur31 ;
```

b) On programme maintenant le contrôle comme suit :

```
package body Controle  is
      S : Semaphore;
      procedure Assurer is begin P(S); end Assurer;
      procedure Relacher is begin V(S); end Relacher
begin
      E0(S,1);
end Controle;
```

Montrer qu'on empêche bien l'interblocage mais pas la famine, si les files d'attente des sémaphores ne sont pas gérées à l'ancienneté, par exemple selon des priorités (supposons Serveur1 plus prioritaire que Serveur2, plus prioritaire que Serveur3, plus prioritaire que Serveur12, plus prioritaire que Serveur23, plus prioritaire que Serveur31)

c) On change l'initialisation du sémaphore en E0(S,2) ; montrer qu'on évite encore l'interblocage. Évite-t-on la famine?

D. ACCÈS DISQUES

On veut comparer plusieurs gestions des accès disques. On utilise des requêtes constituées d'un accès disque "source", d'un traitement suivi d'un accès disque"puit". On suppose pour faire cette comparaison :

- que tous les traitements prennent le même temps de calcul, t,
- que toutes les images "source" sont sur un premier disque
- et que toutes les images "puit" sont sur un deuxième disque,
- que les adresses sur disque sont des adresses de cylindre,
- et que les temps significatifs sont les temps de déplacement des bras porte-têtes sur disque.

Pour faire les comparaisons, on utilise le lot de requêtes suivantes rangées par numéros :

numéro	disque source, initialement bras sur le cylindre 80	traitement de durée	disque puit, initialement bras sur le cylindre 80
1	(70	t	10)
2	(00	t	20)
3	(50	t	70)
4	(90	t	80)
5	(40	t	40)
6	(60	t	90)

Pour ces comparaisons, le serveur est en monoprogrammation. On sert donc les requêtes en séquence selon diverses politiques d'accès aux disques. Dans chaque cas, la position initiale du bras de chaque disque est sur le cylindre 80.

On vous demande

- de réorganiser la séquence des requêtes selon l'ordre de passage que détermine la politique utilisée,
- et de calculer

a) la distance de déplacement totale du bras du premier disque,

b) la distance de déplacement totale du bras du second disque,

c) la somme des deux.

Pour réorganiser la séquence des requêtes on pourra utiliser un tableau comme suit (ici elles sont rangées selon leur numérotation, pour certaines politiques, il faudra les ranger autrement)

numéro de la requête	position du disque source, initiale 80	déplacements du bras du disque source	position du disque puit, initiale 80	déplacements du bras du disque puit
1	70	10	10	70
2	0		20	
3	50		70	
4	90		80	
5	40		40	
6	60		90	
total				

Première politique. On sert les requêtes en séquence selon la numérotation des requêtes (cela correspond soit à leur ordre d'arrivée, soit à une priorité prédéfinie)

Deuxième politique. On sert les requêtes en séquence en appliquant la politique de l'ascenseur au disque "source" (le service commence en montant, à partir du cylindre 80).

Troisième politique. On sert les requêtes en séquence en appliquant la politique de l'ascenseur au disque "puit"(le service commence en montant, à partir du cylindre 80).

Quatrième politique. La prochaine requête à servir est celle qui demande le plus petit déplacement total (déplacement sur le disque "source" + déplacement sur le disque "puit") depuis la requête courante. Avec cette politique, on commence par servir la requête n° 4 (total 10 depuis les positions initiales), puis la requête n° 6 (total 40 depuis la requête 4), etc...

QUESTION D

a) En ne comptant que les temps de déplacement des bras, quelles sont les simplifications faites et quelles sont les ordres de grandeur des marges d'erreur dans le cas d'exécution séquentielle des requêtes?

b) Comparer les déplacements occasionnés pour chaque politique. Quelle est la politique qui donne le moins de déplacement des bras pour l'exécution séquentielle du lot des requêtes (donc le plus petit temps d'exécution)?