SYSTÈMES ET RÉSEAUX INFORMATIQUES

COURS B4 : HTO(19339) et ICPJ(21937)

EXAMEN DU 19 JUIN 2004

portant sur l'enseignement des SYSTÈMES INFORMATIQUE

SOLUTIONS

Solution 1.: Ordonnancement

FIFO	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18
A				A	A	A	A	A	A	A	A	A	A	A	A				
B	B	B	B																
C																C	C		
D																		D	
E																			E
File d'attente	A	AC	ACD	CDE	CDE	CDE	CDE	CDE	CDE	CDE	CDE	CDE	CDE	CDE	CDE	DE	DE	E	

tourniquet	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18
A			A	A						A	A	A	A	A	A	A	A	A	A
B	B	B					B												
C					C	C													
D								D											
E									E										
File d'attente	A	AC	CBD	CBDE	BDEA	BDEA	DEA	EA	A										

EDF	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18
A		A	A	A		A	A	A	A	A	A	A	A	A					
B	B																B	B	
C															C	C			
D																			D
E					E														
File d'attente	A	CB	CBD	ECBD	ACBD	CBD	CBD	CBD	CBD	CBD	CBD	CBD	CBD	CBD	BD	BD	D	D	

Une * indique les cas de faute temporelle

Processus	Durée d'exécution	date de déclenchement	Date d'échéance	FIFO	Quantum	EDF
A	12	1	16	15	19 *	14
B	3	0	17	3	7	18 *
C	2	2	16	17 *	6	16
D	1	3	17	18 *	8	19 *
E	1	4	15	19 *	9	5

Solution 2.1. Synchro

```
-- selon cours 09 Illustration page 6
package body Synchro is
   Sem_Signal : array(Id_Usine) of Semaphore;
   procedure Attendre(X : in Id_Usine) is begin P(Sem_Signal(X)); end Attendre;
   procedure Finir(X : in Id_Usine) is begin V(Sem_Signal(X+1)) ;end Finir;
begin
   for i in Id_Usine loop E0(Sem_Signal(i), 0); end loop; V(Sem_Signal(0));
end Synchro;
```

Solution 2.2. Depot

```
-- selon cours 09 Illustration page 10
package body Depot is
   Nvide, Nplein, Mutex_Cons : Semaphore;
   type Id_Depot is mod 8;   -- type modulo 8 prenant les valeurs de 0 à 7
   T : array(Id_Depot) of Rapport; Tete, Queue : Id_Depot := Id_Depot'First;
   procedure Envoyer(X : in Rapport) is
   begin     -- il n'y a qu'un seul proxy producteur à la fois, pas de concurrence pour le dépôt
      P(Nvide); T(Queue) := X; Queue := Queue + 1; V(Nplein);
   end Envoyer;
   procedure Retirer(X : out Rapport) is
   begin
      P(Nvide);
      P(Mutex_Cons); X := T(Tete); Tete := Tete + 1; V(Mutex_Cons);
      V(Nvide);
   end Retirer;
begin
   E0(Nvide, 8); E0(Nplein, 0); E0(Mutex_Cons, 1);
end Depot;
```

Solution 2.3. Allocateur

Solution 2.3.a. autorisations obtenues une à une

On note que les 3 serveurs, qui ne sont pas synchronisés entre eux, peuvent retirer chacun un rapport pour l'usine 4. Par exemple le premier traite l'usine 4. Pendant ce temps le second traite l'usine 0, puis l'usine 2, puis commence à traiter l'usine 4 pour le rapport suivant de cette usine. Le troisième serveur traite l'usine 1 puis l'usine 3, puis l'usine 0, puis l'usine 1, puis l'usine 2, puis l'usine 3, puis l'usine 4 pour un nouveau rapport. Chaque serveur traite donc un rapport concernant des dates successives de l'usine 4. Cela conduit à demander 4*3 = 12 pages à l'allocateur.

```
-- selon cours 09 Illustration pages 4 et 5
package body Allocateur is
   -- dans ce cas, les autorisations sont allouées une après l'autre
   -- et sont capitalisées par chaque demandeur.
   -- Le pire cas est celui où chacun des 3 serveurs a demandé X=4 pages et a reçu 3 autorisations,
   -- ce qui fait 9 pages déjà réservées.
   -- S'il n'y a que 9 pages, on est en interblocage. Avec 10 au minimum, on l'évite.
   Min : constant := 10;
   N_Allouable: Semaphore;    -- à initialiser avec le nombre des ressources allouables
   procedure Autoriser(I : in Id_S ; X : in Integer) is
      Encore : Integer;
```

```
   begin
      Encore := X; -- sauf erreur, X est positif
      -- demande X autorisations l'une après l'autre
      loop P(N_Allouable); Encore := Encore – 1; exit when Encore = 0; end loop;
   end Autoriser;
   procedure Rendre(I : in Id_S ; X : out Integer)  is
      Encore : Integer;
   begin
      Encore := X; -- sauf erreur, X est positif
      -- retourne X autorisations
      loop V(N_Allouable); Encore := Encore – 1; exit when Encore = 0; end loop;
   end Rendre;
begin
   E0(N_Allouable, Min);
end Allocateur;
```

Solution 2.3.b. autorisations obtenues globalement

```
-- selon cours 09 Illustration page 22
package body Allocateur is
   -- Les autorisations sont allouées globalement. le serveur consomme X autorisations ou zéro.
   -- Avec 4 pages minimum, on peut servir tout serveur demandant 4 pages au plus.
   -- On évite l'interblocage grâce à l'allocation globale.
   Min : constant := 4;
   -- il faut une liste pour stocker les requêtes en attente
   L : FileDe(Id_S, Integer) ;   -- objet fourni avec des procédures d'accès
   Sempriv : array(Id_S) of Semaphore; Mutex : Semaphore;
   Stock : Integer := Min; -- valeur initiale
   procedure Autoriser(I : Id_S ; X : in Integer) is
      Ok : Boolean;              -- sert au schéma des sémaphores privés
   begin
      P(Mutex);
      if X <= Stock then
         Stock := Stock – X;
         Ok := True;
      else
         L.Ajouter(I, X) ; -- on met la requête dans la file d'attente L
         Ok := False ;
      end if;
      V(Mutex);
      if not Ok then P(Sempriv(Id_Bloc)); end if;
   end Autoriser;
   procedure Rendre(I :Id_S ; X : out Integer)  is
      Lui : Id_S ; K : Integer ;
   begin
      P(Mutex) ;
      Stock := Stock + X;
      while not L.EstVide  loop – on sert autant de requêtes que possible en séquence
         L.Premier(Lui, K) ; -- on lit le premier élément de L
         exit when K > Stock ; -- on ne peut pas le servir, c'est fini pour cette restitution
         L.OterPremier ; -- on retire le premier élément de L, pour le servir
         Stock := Stock – K ; sa requête est de K pages
```

```
        V(Sempriv(Lui)) ; -- on autorise lui à prendre ses K pages
      end loop ;
      V(Mutex)
   end Rendre;
begin
   L.Init ; -- initialisation file vide
   E0(Mutex, 1) :
   For I in Id_S loop E0(Sempriv(I), 0) ; end loop ;
end Allocateur;
```

Autre solution 2.3.b. autorisations obtenues globalement

```
-- n'est pas dans le cours de cette année, mais on la trouve avec un objet protégé dans ACCOV
package body Allocateur is
   -- Les autorisations sont allouées globalement. le serveur consomme X autorisations ou zéro.
   -- Avec 4 pages minimum, on peut servir tout serveur demandant 4 pages au plus.
   -- On évite l'interblocage grâce à l'allocation globale.
   Min : constant := 4;
   -- on sert une requête après l'autre et si une requête n'est pas servie, on n'en examine pas d'autre
   -- un sémaphore MutexAutoriser permet de bloquer les nouvelles requêtes quand on en sert une
   -- il ne faut pas de liste pour stocker les requêtes en attente, car il n'y en a qu'une en attente
   -- on ne bloque qu'une demande en cours, donc un seul sémaphore « privé » suffit, SemClient
   MutexAutoriser, SemClient, Mutex : Semaphore;
   Stock : Integer := Min; -- valeur initiale
   Attente : Boolean := False ; -- indique s'il y a une requête en cours qui est en attente
   procedure Autoriser(I : Id_S ; X : in Integer) is
      Ok : Boolean;              -- sert au schéma des sémaphores privés
   begin
      P(MutexAutoriser) ; -- bloque les requêtes tant que la requête en cours n'est pas satisfaite
      P(Mutex);              -- exclusion mutuelle à la variable d'état Stock
      Stock := Stock – X ;
      if Stock >= 0 then
         Ok := True; Attente := False ;
      else
         Ok := False ; Attente := True ;
      end if;
      V(Mutex);
      if not Ok then P(SemClient); end if;
      V(MutexAutoriser) ; -- le client a été servi, on peut traiter une nouvelle requête
   end Autoriser;
   procedure Rendre(I :Id_S ; X : out Integer)  is
   begin
      P(Mutex) ;
      Stock := Stock + X;
      if Attente and Stock >= 0 then
         Attente := False ; V(SemClient) ; -- on autorise le client à prendre les pages demandées
      end if;
      V(Mutex)
   end Rendre;
begin
   E0(MutexAutoriser, 1) ; E0(SemClient, 0) ; E0(Mutex, 1) :
end Allocateur;
```

Solutions 2.4. Droit

Chacun des trois serveurs prend un rapport dans le Dépôt et le traite à son rythme. On peut avoir Serveur(0) qui traite un rapport pour l'usine 4 et que ce soit très long. Pendant ce temps, les Serveur(1) et Serveur(2) traitent les rapports suivants du Dépôt, soit ceux des usines 0, 1, 2, 3. Supposons encore que Serveur(1) prenne ensuite le rapport suivant de l'usine 4 et que ce soit aussi long à traiter. Pendant ce temps Serveur(2) obtient les rapports suivants du Dépôt, ceux des usines 0, 1, 2, 3 et finit par avoir un troisième rapport de l'usine 4. Dans ce cas on aurait les trois Serveurs occupés à traiter trois rapports successifs concernant l'usine 4.
Comme ces trois Serveurs peuvent terminer leurs traitements pour l'usine 4 à tout moment et dans n'importe quel ordre, l'image qui sera visualisée n'est pas toujours la plus récente. Si on voulait que ce soit le cas, il faudrait éviter d'écraser une image par une image plus ancienne et utiliser la date de l'image pour cela.

```
-- Première solution selon cours 09 Illustration pages 13 à 15
package body Droit is
   Mutex : array(Id_Usine) of Semaphore ;
   Mutex_L, Mutex_A : semaphore ;
   NL : Natural := 0 ;
   procedure Debut_Acces_Partiel(X : in Id_Usine) is
   begin
      P(Mutex(X)) ; -- un seul accès pour l'usine X
      P(Mutex_L) ; -- accès concurrents avec des accès pour d'autres usines que X
      NL := NL + 1 ; if NL = 1 then P(Mutex_A) ; end if ;
      V(Mutex_L) ;
   end Debut_Acces_Partiel ;
   procedure Fin_Acces_Partiel is
   begin
      P(Mutex_L) ; -- fin d'accès concurrents
      NL := NL - 1 ; if NL = 0 then V(Mutex_A) ; end if ;
      V(Mutex_L) ;
      V(Mutex(X)) ; -- fin d'accès pour l'usine X, un autre accès pour X peut se faire
   end Fin_Acces_Partiel;
   procedure Debut_Acces_ Total is begin P(Mutex_A) ; end Debut_Acces_ Total;
   procedure Fin_Acces_ Total is begin V(Mutex_A) ; end Fin_Acces_ Total;
begin
   for I in Id_Usine loop E0(Mutex(I), 1) ; end loop ;
   E0(Mutex_L, 1) ; E0(Mutex_A, 1) ;
end Droit;
```

```
-- Deuxième solution, chaque image est considérée comme une ressource en exclusion mutuelle
-- Chaque Serveur ne demande qu'une ressource, le Collecteur les demande toutes
package body Droit is
   Mutex : array(Id_Usine) of Semaphore ;
   procedure Debut_Acces_Partiel(X : in Id_Usine) is
   begin
      P(Mutex(X)) ; -- un seul accès pour l'usine X
   end Debut_Acces_Partiel ;
   procedure Fin_Acces_Partiel is
   begin
      V(Mutex(X)) ; -- fin d'accès pour l'usine X, un autre accès pour X peut se faire
   end Fin_Acces_Partiel;
```

```
   procedure Debut_Acces_ Total is
   begin
   for I in Id_Usine loop P(Mutex(I)) ; end loop ;  -- demande l'accès à toutes les images
   end Debut_Acces_ Total;
   procedure Fin_Acces_ Total is
   begin
   for I in Id_Usine loop V(Mutex(I)) ; end loop ;  -- libère l'accès à toutes les images
   end Fin_Acces_ Total;
begin
   for I in Id_Usine loop E0(Mutex(I), 1) ; end loop ;
end Droit;

-- Il n'y a pas d'interblocage, par contre le Collecteur peut attendre indéfiniment
-- si les Serveurs peuvent se coaliser pour monopoliser l'image de l'usine X
-- en étant prioritaires dans la file d'attente du sémaphore Mutex(X)
```