Université Bordeaux I
Session de janvier 2001

U.F.R. Mathématiques et Informatique
Alain Griffault

Maîtrise d'Informatique
Systèmes d'exploitation
(Durée : 2 heures)

Documents interdits

1 Cours : 2 points

Question 1.1 *Décrivez (avec un schéma éventuellement) l'exécution d'un appel de procédure distant (RPC).*

2 Algorithme du banquier : 4 points

Un système est composé de trois processus P_1, P_2, P_3 et de quatre types de ressource.

Initialement, le système est décrit par $Existe = (5, 6, 6, 5)$, et $Max = \begin{pmatrix} 4 & 3 & 3 & 1 \\ 2 & 3 & 3 & 4 \\ 2 & 2 & 3 & 3 \end{pmatrix}$.

Soit la situation $Tenue = \begin{pmatrix} 2 & 1 & 2 & 0 \\ 1 & 1 & 2 & 1 \\ 1 & 2 & 1 & 1 \end{pmatrix}$.

Question 2.1 *La situation est-elle saine ? Justifiez.*

Question 2.2 *On suppose que le processus P_1 émet la requête $R_1 = (0, 1, 1, 0)$. La requête va t'elle être honorée de suite, ou bien va t'elle être mise en attente ? Justifiez.*

3 Algorithme de remplacement de pages : 6 points

L'idée de cet algorithme est d'évincer la page la moins fréquemment utilisée. Pour cela cet algorithme associe à chaque cadre un compteur qui comptabilise le nombre d'accès à la page contenue dans ce cadre.

Question 3.1 *Définissez un algorithme basé sur cette idée. Vous devez décrire précisément :*

1. *la valeur initiale des compteurs,*
2. *les événements qui provoquent les modifications (incrémentation, décrémentation, affectation...),*
3. *la fonction de sélection de la page à évincer.*

Question 3.2 *Construisez, pour une mémoire comportant quatre cadres initialement vides, la suite des configurations de la mémoire, pour la suite de références :*
`1, 2, 3, 4, 5, 3, 4, 1, 6, 7, 8, 7, 8, 9, 7, 8, 9, 5, 4, 5, 4, 2.`

Question 3.3 *Donnez une suite infinie de références qui après avoir chargée une page dans le premier cadre, ne provoquera plus jamais de défaut de page sur ce cadre, bien que la page ne soit pas infiniment souvent utilisée. Donnez une interprétation possible à votre suite de références.*

Question 3.4 *En utilisant une technique de type* vieillissement*, décrivez les modifications à apporter à votre algorithme afin de corriger le problème précédent.*

4 Moniteurs et sémaphores : 8 points

Definition 4.1 *Un sémaphore* `S` *est une structure de données*

`type semaphore = (n : entier; l : liste de processus);`

manipulée uniquement par les actions atomiques `wait(S)` *et* `signal(S)`*, également nommées* `P(S)` *et* `V(S)`*.*

```
procedure wait(var S : semaphore)
debut
S.n := S.n - 1;
si (S.n < 0) alors debut
        ajouter(moi,S.l);
        block;
        fin
fin
```

```
procedure signal(var S : semaphore)
debut
S.n := S.n + 1;
si (S.n <= 0) alors debut
        enlever(P,S.l);
        wakeup(P);
        fin
fin
```

Definition 4.2 *Un moniteur est constitué d'un ensemble de variables, d'un ensemble de conditions et d'un ensemble de procédures.*

les variables *sont locales au moniteur, donc non accessibles aux utilisateurs.*

les conditions *sont locales au moniteur, donc non accessibles aux utilisateurs. Une condition* `c` *n'est manipulable qu'au moyen de deux primitives :*
- `c.attendre` *: bloque le processus* `P` *et le place en attente de* `c`*,*
- `c.signaler` *: réveille un processus en attente de* `c` *s'il en existe au moins un, ne fait rien sinon.*

les procédures *pour un moniteur* `M` *sont externes ou bien locales.*
- *les procédures externes sont celles qui sont accessibles aux utilisateurs. Un moniteur garantit que ses procédures externes sont exécutées en exclusion mutuelle.*
- *les procédures locales sont :* `init(M)`, `entree(M)`, `sortie(M)`*. La procédure locale* `init(M)` *est exécutée une seule fois à la création du moniteur. La procédure* `entree(M)` *(resp.* `sortie(M)`*) est exécutée au début (resp. à la fin) de chaque procédure externe. Elles y sont ajoutées par le compilateur, et ce sont elles (avec les primitives sur les conditions) qui font l'implémentation du moniteur.*

Comme exemple, le moniteur suivant protège la lecture d'une variable `a` non initialisée.

```
sync : moniteur;                      |  processus P
  var fait : boolean;                 |    debut
      fini : condition;               |    ecrire(a);
procedure finEcrire;                  |    sync.finEcrire;
  debut                               |    fin
  fait := vrai;                       |
  fini.signaler                       |  processus Q
  fin                                 |    debut
procedure debutLire;                  |    sync.debutLire;
  debut                               |    lire(a);
  si (non fait) alors                 |    fin
    fini.attendre
  fin
procedure init;
  debut
  fait := faux
  fin
```

Question 4.1 *Donnez le code des procédures externes* `debutEcrire` *et* `finEcrire` *du moniteur suivant qui procure une solution aux problèmes des lecteurs rédacteurs. La priorité est donnée aux lecteurs, une écriture n'est pas autorisée si des lecteurs sont en attente.*

```
fichier : moniteur;
  var : ecrire : booleen;
        nbLecteurs : entier;
        cEcrire, cLire : condition;
procedure debutLire;                 | processus lecteur;
  debut                              |   debut
  nbLecteurs := nbLecteurs + 1;      |   fichier.debutLire;
  si ecrire alors                    |   < accès aux données protégées >
     debut                           |   fichier.finLire;
     cLire.attendre;                 |   fin
     cLire.signaler;                 |
     fin                             |
  fin                                | processus redacteur;
procedure finLire;                   |   debut
  debut                              |   fichier.debutEcrire;
  nbLecteurs := nbLecteurs - 1;      |   < accès aux données protégées >
  si (nbLecteurs = 0) alors          |   fichier.finEcrire;
     cEcrire.signaler                |   fin
  fin
procedure init;
  debut
  ecrire := faux;
  nbLecteurs := 0;
  fin
```

On se propose maintenant d'implémenter les moniteurs à partir des sémaphores.

Question 4.2 *Dans la cas ou la primitive* `signaler` *n'est admise que comme dernière instruction d'une procédure externe, donnez une implémentation : d'une part des primitives* `signaler` *et* `attendre` *pour une condition ; d'autre part des procédures locales* `init(M)`, `entree(M)`, `sortie(M)`. *Vous pourrez utiliser :*
– *un sémaphore pour gérer l'exclusion entre les procédures externes,*
– *pour chaque condition, un sémaphore et un compteur du nombre de processus en attente.*

Dans la cas ou la primitive `signaler` est admise partout dans une procédure externe, se pose le problème suivant : Lorsque `P` exécute l'instruction `c.signaler`, cela réveille un processus `Q` ; il y a alors deux processus actifs en même temps dans deux procédures externes du moniteur. Il faut donc en endormir un des deux, il existe deux possibilités :

1. `P` attend que `Q` abandonne le moniteur ou attend une autre condition.
2. `Q` attend que `P` abandonne le moniteur ou attend une autre condition.

Question 4.3 *Pour la sémantique que vous préciserez, donnez une implémentation : d'une part des primitives* `signaler` *et* `attendre` *pour une condition ; d'autre part des procédures locales* `init(M)`, `entree(M)`, `sortie(M)`. *Vous pourrez utiliser :*
– *un sémaphore pour gérer l'exclusion entre les procédures externes,*
– *un sémaphore et un compteur pour gérer la priorité entre* `P` *et* `Q`*,*
– *pour chaque condition, un sémaphore et un compteur du nombre de processus en attente.*