Université BORDEAUX I
Session de septembre 1997

U.F.R. Mathématiques et Informatique
Alain Griffault

Maîtrise d'Informatique
Systèmes d'exploitation 1
(Durée: 2 heures)

Documents interdits.

1 Cours : 4 points

1. Quel est le rôle de l'attribut *compteur de référence* contenu dans l'I-node d'un fichier Unix ?
2. Dans le système Unix, un processus qui attend la terminaison d'une entrée/sortie sur disque est plus prioritaire qu'un processus qui attend qu'un tampon soit disponible. Expliquer brièvement.

2 Algorithmes de remplacements de pages : 6 points

1. Construire, pour une mémoire comportant 3 cadres initialement vides, des suites de références les plus simples (courtes) possibles pour lesquelles :
   (a) le nombre de défauts de pages est plus petit pour l'algorithme FIFO que pour LRU.
   (b) le nombre de défauts de pages est plus petit pour l'algorithme LRU que pour FIFO.
2. Construire une suite de références la plus simple possible qui met en évidence l'anomalie de Belady : **pour l'algorithme FIFO, l'augmentation de la mémoire peut augmenter le nombre de défauts de pages**. Vous pourrez chercher à continuer la suite **0 1 2 3 0 1 4** sur une mémoire constituée de 3 pages, puis de 4 pages.

3 Synchronisation de processus : 10 points

N.B. : les questions 1, 2 et 3 sont indépendantes.

N philosophes et N fourchettes sont placés alternativement autour d'une table circulaire. Un philosophe ne peut manger que s'il possède les deux fourchettes qui lui sont attribuées; c'est à dire qu'un philosophe ne peut manger que si aucun de ses voisins (immédiats) ne le fait. Les solutions à ce problème doivent éviter **les blocages** (tous les processus philosophes sont *en attente*), **la famine généralisée** (des processus philosophes sont de temps en temps *actifs*, mais aucun ne parvient à manger), et être **équitable** (un philosophe qui le désire n'attendra pas indéfiniment avant de manger).

1. La solution suivante pour le philosophe i est la plus naturelle. Les fourchettes sont des objets non partageables et sont représentées par des sémaphores.

```
 fourchette : tableau[0..(N-1)] de sémaphores initialisés à 1 ;

tant que vrai faire
              pense;
              P(fourchette[i]);
              P(fourchette[(i+1)modulo N]);
              mange;
              V(fourchette[i]);
              V(fourchette[(i+1)modulo N]);
              fait
```

   (a) Ajouter des commentaires au programme précédent afin d'expliquer son fonctionnement.
   (b) Donner un comportement des philosophes ($N = 4$) qui montre que cette solution n'est pas exempte de blocages.

2. La solution suivante cherche à résoudre le problème précédent en utilisant les structures de données suivantes :

```
un sémaphore mutex initialisé à 1.
philo : tableau[0..(N-1)] de sémaphores initialisés à 0 ;
etat : tableau[0..(N-1)] de (pense,mange,affame);
```

Le code des processus philosophes est alors :

```
tant que vrai faire
              pense;
              prendre_couverts(i);
              mange;
              poser_couverts(i);
              fait
```

avec les procédures suivantes :

```
prendre_couverts(i);
   debut
   P(mutex);
   etat[i] := affame;
   test(i);
   V(mutex);
   P(philo[i]);
   fin

poser_couverts(i);
   debut
   P(mutex);
   etat[i] := pense;
   test((i-1) modulo N);
   test((i+1) modulo N);
   V(mutex);
   fin

test(i)
   debut
   si (etat[i]=affame) et (etat[(i-1) modulo N]<>mange) et (etat[(i+1) modulo N]<>mange) alors
       debut
       etat[i] := mange;
       V(philo[i]);
       fin
   fin
```

   (a) Ajouter des commentaires au programme précédent afin d'expliquer son fonctionnement.

   (b) Donner un comportement des philosophes ($N = 4$) qui montre que cette solution n'est pas équitable.

3. Soit les structures de données :

```
un sémaphore mutex initialisé à N-1.
fourchette : tableau[0..(N-1)] de sémaphores initialisés à 1 ;
```

   (a) Trouver une solution correcte au problème précédent en utilisant ces structures de données.

   (b) Ajouter y des commentaires afin que je comprenne son fonctionnement.